Neural Networks and Complexity Theory

Pekka Orponen

Department of Computer Science, University of Helsinki
Teollisuuskatu 23, SF–00510 Helsinki, Finland
E-mail: orponen@cs.helsinki.fi

Abstract. We survey some of the central results in the complexity theory of discrete neural networks, with pointers to the literature.

1 Introduction

The recently revived field of computation by "neural" networks provides the complexity theorist with a wealth of fascinating research topics. While much of the general appeal of the field stems not so much from new computational possibilities, but from the possibility of "learning", or synthesizing networks directly from examples of their desired input-output behavior, it is nevertheless important to pay attention also to the complexity issues: firstly, what kinds of functions are computable by networks of a given type and size, and secondly, what is the complexity of the synthesis problems considered. In fact, inattention to these issues was a significant factor in the demise of the first stage of neural networks research in the late 60's, under the criticism of Minsky and Papert [51].

The intent of this paper is to survey some of the central results in the complexity theory of neural network computation, as developed to date. We give no proofs. The paper might be most profitably read in conjunction with Ian Parberry's earlier survey [57], which also gives the proofs of many of the most significant (semi-) elementary results. Our emphasis is at some points slightly different from Parberry's, and we also update his survey with some of the more recent developments. Other surveys of related topics, partially overlapping the present one, are [58, 72]. As a general introduction to neural networks theory, although mostly other than complexity aspects, the excellent textbook [33] can be recommended, and certain aspects of computations in cyclic networks are covered in depth in [40]. The original "PDP" books [67, 48] still make very inspiring reading, and many significant original papers have been reprinted in the anthologies [4, 5]. Also, earlier texts on threshold logic such as the comprehensive [52] contain a wealth of material that has again become very relevant.

Finally, a caveat: this brief survey necessarily ignores many aspects of the issues covered (e.g., average case vs. worst case results, different models of learning, etc.) For further details, see the works listed in the Bibliography, and the references therein.

2 Preliminaries

The neural networks literature abounds with seemingly very different neural models of computation, intended for different kinds of applications and with different synthesis algorithms (learning rules). Luckily, from the point of view of their computational capabilities, the number of fundamentally different models is far fewer.

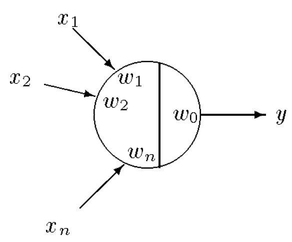

Fig. 1. A formal neuron.

We may vaguely define a *neural network* to be any network-like model of computation whose basic computational unit is some kind of a *formal neuron*, as illustrated in Fig. 1. A neuron receives input signals $x_1, \ldots, x_n$ from either other neurons or the outside environment. It computes its output signal y by adding together the x_j's weighted by some internal *weights* w_j, possibly subtracting a *bias* or *threshold* term w_0, and applying some *transfer function* σ to the result:

$$y = \sigma(\sum_{j=1}^{n} w_j x_j - w_0).$$

For simplicity and uniformity, we shall usually not distinguish between the bias terms and the other weights. There are also more complicated models, where either the combination of the input signals is not a simple summation, or the neurons may have more complicated internal states, but we shall not consider those models here. (In particular, we are ignoring the widely researched models of *cellular automata* and *automata networks*; for uniform treatments of neural and automata networks see [13, 24]).

In the most general case, all the parameters of the model – the weights and the input and output signals – may be arbitrary real numbers, but they may also be restricted to integral values, to reals or integers from within some interval $[-M, \ldots, M]$ (as in, e.g, [55]), or often just to integers from $\{-1, 1\}$, $\{0, 1\}$, or $\{-1, 0, 1\}$.

The transfer function σ may be either a strict threshold function:

$$\mathrm{sgn}(t) = \begin{cases} 0 \text{ if } t < 0, \\ 1 \text{ if } t \geq 0, \end{cases}$$

or a smooth "sigmoid" such as

$$\sigma(t) = (1 + e^{-t})^{-1},$$

or possibly a piecewise linear approximation of the latter. (These forms of the functions can be used when the output signals are restricted to the interval $[0, 1]$. For different output intervals, the transfer functions must be adjusted accordingly.) The transfer function may even be stochastic, as in the *Boltzmann machine* model [34, 1], where its behavior moreover depends on a time-dependent "temperature" parameter T:

$$\Pr(\sigma_T(t) = 1) = (1 + e^{-t/T})^{-1}.$$

An important special type of neuron is the *threshold gate*, with 0/1 input and output signals, all weights equal to 1 except the bias term, which may be an arbitrary integer, and a threshold transfer function.

A network can be either *cyclic* or *acyclic*, and the interconnections between the neurons in a cyclic network may be either *symmetric* or *asymmetric*. Denoting the weight given at neuron i for the input signal coming from neuron j by w_{ij}, symmetricity means that $w_{ij} = w_{ji}$, for all neurons i, j. A cyclic network is *simple* if $w_{ii} = 0$ for all neurons i.

In a cyclic network, it is customary to call the neuron output signals their *states*. In such a network, the update method for the states is of significance. The update may be *continuous*, in which case the behavior of the network is described by a set of differential equations, or it may be *discrete*, in which case the differential equations are replaced by iterated functions. We shall here consider only the discrete-time models, although the continuous ones are also of considerable importance (for references, see [33]).

In a discrete-time cyclic network the updates may be *synchronous*, in which case all the neurons are updated simultaneously in parallel, or *asynchronous*, in which case the neurons are selected for updating one at a time in some order. A global state of a cyclic network is *stable* if none of the neurons would change its state in an update. The collection of stable states of a network is independent of whether the updates are performed synchronously or asynchronously.

Some well-known neural network models are the following (for more information on each, see [33, 67]):

The perceptron. A single neuron with real-valued input signals and weights, a 0/1 output signal, and a threshold transfer function. A Boolean-valued function defined on some set of points in the input space is called *linearly separable* if it can be computed by a single perceptron. (The name derives from the fact that a perceptron basically implements a hyperplane in the input space, dividing the points with output 0 from the points with output 1.)

The backpropagation network. An acyclic network of neurons, usually of bounded depth. In this model, typically all the parameters are arbitrary real numbers, and the transfer function is a sigmoid such as $\sigma(t) = (1 + e^{-t})^{-1}$.

The Hopfield net. A symmetric, simple, fully connected cyclic network. Typically discrete-time, with neuron states 0/1 or -1/1, and a threshold transfer function; but also the continuous-time version, with real-valued states and a sigmoid transfer function, appears in the literature.

The Boltzmann machine. A stochastic version of the discrete-time Hopfield net, with transfer function σ_T, where T is lowered to 0 during computation.

3 Acyclic nets

In the neural networks literature, the most powerful and commonly studied acyclic network model is the backpropagation net, with arbitrary real number parameters and a sigmoid transfer function. At the other end of the scale is the *threshold circuit* where all the neurons are simple threshold gates. An intermediate model we shall consider is the *neural circuit* or "multilayer perceptron", where the weights may be arbitrary reals, but the input and output signals are discretized to 0/1 values by threshold transfer functions.

3.1 Computational power

We shall concentrate on using acyclic networks to compute (sequences of) single-valued Boolean functions, i.e., mappings from $\{0,1\}^n$ to $\{0,1\}$, for growing values of n. Since threshold circuits can strictly speaking only compute monotone functions, we shall asssume that a network gets as input both the actual input bits $x_1, \ldots, x_n$ and their negations $\bar{x}_1, \ldots, \bar{x}_n$.

In comparing different networks computing a given function, the important parameters to consider are the *size*, *depth* and *weight* of a network, defined respectively as the number of neurons, the length of a longest path from an input point to an output neuron, and the sum total of the absolute values of all the weights in the network. Because real number parameters may be scaled at will, the weight measure really makes sense only for networks with integer parameters.

The following result is fundamental ([53, 52]; for recent versions of the proof, see [31, 35, 57, 58, 62]):

Theorem 1. *Any linearly separable Boolean function on n variables can be implemented by a perceptron with integer weights w_i such that $|w_i| \leq (n+1)^{(n+1)/2}/2^n$, for all $i = 0, \ldots, n$.*

A converse to this was proved only very recently [31] (although weaker versions of the converse have been known since the early 60's, cf. [52, 54, 57]):

Theorem 2. *For infinitely many n, there are linearly separable Boolean functions on n variables that require weights as large as $n^{n/2}/2^n$ in a single perceptron implementation.*

These theorems imply that polynomial size neural circuits may be divided in two subclasses: circuits whose weights are polynomially bounded, and circuits whose weights are not polynomially bounded, but nevertheless are representable in $O(s \log s)$ bits each, where s is the size of the circuit. A polynomial size neural circuit with small weights can be easily implemented as a polynomial size threshold circuit of the same depth [57], and a polynomial size neural circuit with large weights can be implemented as a polynomial size threshold circuit of at most twice the depth [21].

In analogy to the standard AND/OR/NOT circuit complexity classes AC^k and NC^k [71], the following neural, or equivalently threshold circuit complexity classes are defined for each $k \geq 0$:

$$\mathrm{TC}^k = \{\text{functions computable by neural circuits of polynomial size and depth } O(\log^k n)\}.$$

It is quite easy to show that for all $k \geq 0$,

$$\mathrm{AC}^k \subseteq \mathrm{TC}^k \subseteq \mathrm{NC}^{k+1}.$$

Of these inclusions, only the inclusion of AC^0 in TC^0 is known to be proper. This separation is witnessed by, e.g., the majority function, which is known not to be in AC^0 [17, 73, 30, 71], but can of course be computed by a single threshold gate. It is at the moment quite conceivable that $\mathrm{TC}^0 = \mathrm{NC}^1$, although the general conjecture seems to be the opposite.

Polynomial size, bounded depth threshold circuits are surprisingly powerful. All symmetric Boolean functions, as well as the comparison and addition on two n-bit numbers can be computed by such networks in depth 2 [52, 57], the product of two n-bit numbers can be computed in depth 4 [70], and analytic functions with a convergent rational power series (e.g., sin, cos, exp, log, sqrt) can all be approximated in bounded depth [63] (cf. also [69]). It is therefore of great interest to consider also the following fixed-depth sublevels of the class TC^0 individually:

$$\mathrm{TC}_d^0 = \{\text{functions computable by neural circuits of polynomial size and depth } d\},$$

$$\widehat{\mathrm{TC}}_d^0 = \{\text{functions computable by threshold circuits of polynomial size and depth } d\}.$$

(Some authors, e.g. [21, 70], use the notations LT_d, $\widehat{\mathrm{LT}}_d$, from "linear thresholds", for these classes.) It is shown in [21] that the classes $\widehat{\mathrm{TC}}_d^0$, TC_d^0 form a hierarchy:

$$\widehat{\mathrm{TC}}_d^0 \subseteq \mathrm{TC}_d^0 \subseteq \widehat{\mathrm{TC}}_{d+1}^0$$

for all $d \geq 1$. Separating the levels of this hierarchy, as far as they are separate, is currently a major research task.

That class $\widehat{\mathrm{TC}}_1^0$ is properly contained in TC_1^0 follows from Theorem 2 above; and the proper containment of TC_1^0 in $\widehat{\mathrm{TC}}_2^0$ follows from the well-known fact [51] that the parity function cannot be computed by a single perceptron, but being a symmetric function, it can be computed by a small threshold network of depth 2. The classes $\widehat{\mathrm{TC}}_2^0$ and TC_2^0 are separated by a result in [21]. The separation of $\widehat{\mathrm{TC}}_2^0$ from $\widehat{\mathrm{TC}}_3^0$ was first proved in [26]. The separating function is a generalization of parity called "inner product mod 2":

$$ip_2(x_1, \ldots, x_n, y_1, \ldots, y_n) = \bigoplus_{i=1}^{n} (x_i \wedge y_i),$$

where $\oplus$ denotes the "exclusive or" operation. The proof in [26] basically consists in showing that the ip_2 function correlates only weakly with any function computable by a perceptron with polynomially bounded weights, but strong correlation would be necessary for the function to be computable by a small two-layer circuit of threshold gates. (However, the function could still conceivably be computable by a small two-layer circuit of unbounded-weight perceptrons.) This important technique has recently been simplified and used to prove new results in [21, 32, 65].

The separation of $\widehat{\mathrm{TC}}_2^0$ from $\widehat{\mathrm{TC}}_3^0$ stands in contrast to the often-cited results in the neural networks literature [9, 16, 37] proving that two-layer backpropagation nets "are sufficient", in the sense of being capable of approximating arbitrary continuous functions. However, the latter results are essentially analogous to the Boolean normal form theorems asserting the existence of a depth 2 representation for any Boolean function — albeit in almost all cases an exponentially large one [71].

Still, there could in principle be some representational efficiency to be gained from using continuous instead of threshold transfer functions. This question was studied in [47], where it was shown, under a reasonable definition of what it means for a continuous-valued network to compute a Boolean function, and very modest conditions on the continuous, nonlinear transfer function σ, that for all $d \geq 1$,

$$\widehat{\mathrm{TC}}^0_d(\sigma) = \widehat{\mathrm{TC}}^0_d,$$

where $\widehat{\mathrm{TC}}^0_d(\sigma)$ denotes the class of Boolean functions computable by polynomial size, polynomial weight, depth d backpropagation nets with transfer function σ.

Also acyclic nets with stochastic transfer functions can be reduced, by relatively standard complexity theory techniques [59, 57], to the basic deterministic neural circuit model. However, the argument showing the existence of a deterministic circuit sequence corresponding to a sequence of stochastic circuits is nonconstructive, and hence the resulting sequence is potentially nonuniform.

3.2 Synthesis

The possibility of synthesizing neural networks from examples of their input/output behavior is a central motivating factor for the field. Consequently, many algorithms have been developed for solving this problem, beginning with the celebrated "perceptron converge procedure" [64, 51, 33] for single perceptrons, and the more recent "backpropagation algorithm" [66] for backpropagation nets. The problem may be precisely formulated in different ways, depending on how much of the network architecture is given as input, and what precisely qualifies as a solution. The first complexity results regarding the synthesis problem are due to Judd [38, 39], who proved that the following version is NP-complete:

The Loading Problem. Given a set of pairs $\{(\mathbf{x}_1, b_1), \ldots, (\mathbf{x}_m b_m)\}$, where each $\mathbf{x}_i \in \{0,1\}^n$ and each $b_i \in \{0,1\}$, and a directed acyclic graph, is there an assignment of weights to the nodes in the graph, such that for the function f computed by the resulting neural circuit, $f(\mathbf{x}_i) = b_i$ for all $i = 1, \ldots, m$?

Judd's proof depends quite significantly on the fact that also the network architecture is given as part of the input in the problem. The result was improved by Blum and Rivest [7], who showed that the loading problem remains NP-complete even when restricted to the simple three-node network structure presented in Fig. 2.

Still, the Blum/Rivest result leaves open the possibility that it is an artefact of the fixed number of nodes in the architecture, and the way they are connected. Lin and Vitter [46] further proved that the loading problem is NP-complete also for the "cascade" two-node architecture presented in Fig. 3. Since any other two-node network can be obtained from the cascade network by setting some weights to zero, the Lin/Vitter result also implies that the following problem is NP-complete:

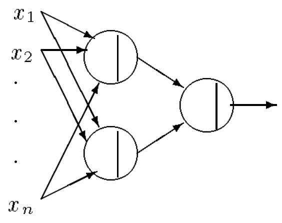

Fig. 2. A three-node network.

Neural Circuit Minimization. Given a set of pairs $\{(\mathbf{x}_1, b_1), \ldots, (\mathbf{x}_m b_m)\}$, where each $\mathbf{x}_i \in \{0,1\}^n$ and each $b_i \in \{0,1\}$, and an integer K, is there a neural circuit of at most K neurons, such that for the function f computed by the circuit, $f(\mathbf{x}_i) = b_i$ for all $i = 1, \ldots, m$?

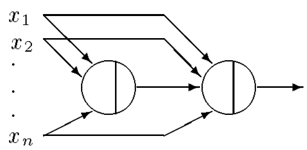

Fig. 3. A "cascade" two-node network.

It should be pointed out that the problem of whether a given set of input/output pairs can be implemented on a *single* neuron can be solved in polynomial time by linear programming techniques. (The requirements for correct loading can be written as a set of linear inequalities, with the weights as unknowns.) However, the original perceptron convergence procedure for solving this problem requires exponential time in the worst case. (This follows from Theorem 2 above, as the procedure changes the weights of a neuron in fixed increments.)

Of course, in practical neural network synthesis, one is typically not interested in finding the absolutely minimal circuit responding correctly to a given set of examples; some circuit not too much larger than the minimum would do just as well. However, Kearns and Valiant [42] have proved that under cryptographic assumptions, it is intractable to find a bounded depth threshold circuit implementing a given set of input/output pairs that is at most polynomially larger than the minimal possible one.

4 Cyclic nets

4.1 Computational power

It has been known since the early work of McCulloch and Pitts [49] and Kleene [43] (see also [50]) that finite asymmetric networks of threshold gates are equivalent to finite automata[1]; and various infinitary analogs of neural networks have recently been shown to be equivalent to Turing machines. For the latter type of result, constructions using a potentially infinite network are presented in [15, 29], and constructions using a finite network, but real-valued neurons of arbitrary precision are presented in [29, 60, 68]. The construction in [68] is rather interesting, as there the required precision grows only linearly in the space requirement of the simulated Turing machine.

In these models the input to a net is given as a sequence of pulses. Another convention, closer to the current applications' point of view, is to load the input initially to a set of designated input neurons, and then let the network run unintervened until it (possibly) stabilizes, at which point the output is read from a set of designated output neurons (in the case of Boolean function computations, a single output neuron). This formalization naturally requires that the network grow with increasing input size. If the only part of the network that changes is the set of input neurons, then there is no difference to the sequential input model; otherwise, the situation is different.

Since a cyclic net of size s stabilizing in time t may be unwound into an acyclic net of size $s \cdot t$, the class of Boolean functions computed by polynomial size, polynomial time cyclic nets coincides with the class P/poly of functions computed by polynomial size acyclic nets. On the other hand, the class of functions computed by polynomial size cyclic nets in *unbounded* time can be seen to equal the class PSPACE/poly considered in [6, 41]. (Parberry in [57] attributes this result to the unpublished report [45].) For an interesting approach to computations on nets of unbounded size, see [14, 15, 18, 19].

Concerning symmetric nets, the fundamental result is Hopfield's [36] observation that a symmetric simple net using an asynchronous update rule always stabilizes. This was improved in [12] to show that in a symmetric simple network of n neurons with integer weights w_{ij}, the stabilization requires at most a total of

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

neuron state changes, under an asynchronous update rule. Under synchronous updates, a similar bound holds [22], but the network may also converge to oscillate between two alternating states instead of a unique stable state (see also [8]). Thus, in particular, networks with polynomially bounded weights stabilize in polynomial time. On the other hand, networks with exponentially large weights may indeed require an exponential time to stabilize, as was first shown in [25] for synchronous

[1] An interesting question here is how efficient is the representation of finite automata as neural nets. It was shown recently in [3] that representing an automaton of n states may require $\Omega((n \log n)^{1/3})$ neurons in the worst case.

updates (a simplified construction appears in [23]), and in [28] for a particular asynchronous update rule. (For a different update rule the latter result actually follows already from [25] or [23] by a fairly simple construction.) Finally, a network requiring exponential time for stabilization under any asynchronous update rule was demonstrated in [27]. Related work appears in [8, 24].

Somewhat surprisingly, the very constrained convergence behavior of symmetric nets is not reflected in their computational power, at least not when the synchronous update rule is used. In this model, polynomial size symmetric nets with unbounded weights are capable of computing all of PSPACE/poly, and polynomial size symmetric nets with polynomially bounded weights are capable of computing all of P/poly [56].

4.2 Synthesis

The most significant application areas for cyclic neural nets proposed so far are associative (or error-correcting) memory, and solving combinatorial optimization problems. Numerous algorithms for synthesizing cyclic neural nets with predetermined stable states have been proposed in the literature, with the most recent spur of activity starting with Hopfield's [36] "outerproduct" rule. We shall not discuss these algorithms here, nor consider the very interesting issue of the *capacity* of various network models as associative memories (i.e., how many stable states, with preferably large attraction basins, can be created in a network of a given size, and what is the best algorithm for doing this). A large literature exists on these topics, and we are content to refer the reader to the in-depth treatment in [40]. (For some of the several alternative points of view, see also [33, 44].)

Instead, we wish to point out the following complexity results on problems arising in the analysis of cyclic networks.

1. It is an NP-hard problem to decide if a given asymmetric network will stabilize from all initial states [61], or from any initial state [2, 20].
2. It is an NP-complete problem to decide if a given symmetric, simple network has more than one stable state [20]. In fact, the problem of counting the number of stable states is #P-complete [11]. (By Hopfield's convergence result, a symmetric simple net always has at least one stable state.)
3. It is an NP-complete problem to determine if a given stable state in a symmetric simple network has a nontrivial attraction basin under synchronous updates (i.e., if the network converges to the given state from any other initial state) [11]. Computing the size of the attraction basin is #P-hard.
4. Computing the attraction radius, under synchronous or asynchronous updates, of a given stable state in a symmetric simple network is NP-hard. (The attraction, or error-correcting radius of a stable state is the maximal Hamming distance from within which all initial states eventually converge to the given stable state.) In fact, the attraction radius cannot be even approximated to within a factor of $n^{1-\epsilon}$, for any $\epsilon > 0$, unless P = NP [10].

5 Open Problems

Among the open problems in the complexity theory of neural networks perhaps the most intriguing ones are those related to the TC hierarchies: is the bounded-depth TC hierarchy infinite, as is the corresponding AC hierarchy [73, 30]? Even the separations of TC_2^0 from $\widehat{\mathrm{TC}}_3^0$, and $\widehat{\mathrm{TC}}_3^0$ from $\widehat{\mathrm{TC}}_4^0$ are open. If the hierarchy is finite, is $\mathrm{TC}^0 = \mathrm{NC}^1$? Or a potentially easier question: is $\mathrm{TC}^1 = \mathrm{NC}^2$? Also, is $\mathrm{AC}^0 \subseteq \mathrm{TC}_d^0$ for some constant d?

From the point of view of developing a unified theory of neural computation, it would be of significance to extend the result of [47] on the equivalence of sigmoid and threshold transfer functions for small-weight networks to arbitrary weights; i.e., to prove that for any reasonably smooth nonlinear transfer function σ,

$$\mathrm{TC}_d^0(\sigma) = \mathrm{TC}_d^0$$

holds for all $d \geq 1$.

An interesting problem of some significance would also be to determine the complexity of the "Hopfield net loading problem": Given a set of vectors $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$, where each $\mathbf{x}_i \in \{0,1\}^n$, and a constant ρ, is there a symmetric net of n neurons that has these vectors as stable states, with attraction radii $\geq \rho$? For $\rho = 0$, the problem is solvable by linear programming, but it is apparently unknown whether it is in P or NP-complete for any nontrivial $\rho \geq 1$. Of course, all the network synthesis algorithms proposed in the literature provide in some sense approximate solutions to this problem.

Acknowledgment

I am grateful to Mikael Goldmann for pointing out an error in an earlier version of this paper.

References

1. Aarts, E., Korst, J. *Simulated Annealing and Boltzmann Machines.* John Wiley & Sons, Chichester, 1989.
2. Alon, N. Asynchronous threshold networks. *Graphs and Combinatorics 1* (1985), 305–310.
3. Alon, N., Dewdney, A. K., Ott, T. J. Efficient simulation of finite automata by neural nets. *J. Assoc. Comp. Mach. 38* (1991), 495–514.
4. Anderson, J. A., Rosenfeld, E. (eds.) *Neurocomputing: Foundations of Research.* The MIT Press, Cambridge, MA, 1988.
5. Anderson, J. A., Pellionisz, A., Rosenfeld, E. (eds.) *Neurocomputing 2: Directions for Research.* The MIT Press, Cambridge, MA, 1991.
6. Balcázar, J. L., Díaz, J., Gabarró, J. On characterizations of the class PSPACE/poly. *Theoret. Comput. Sci. 52* (1987), 251–267.
7. Blum, A. L., Rivest, R. L. Training a 3-node neural network in NP-complete. *Neural Networks 5* (1992), 117–127.
8. Bruck, J. On the convergence properties of the Hopfield model. *Proc. of the IEEE 78* (1990), 1579–1585.

9. Cybenko, G. Approximation by superposition of a sigmoidal function. *Math. of Control, Signals, and Systems 2* (1989), 303–314.
10. Floréen, P., Orponen, P. Attraction radii in binary Hopfield nets are hard to compute. Manuscript submitted for publication, 7 pp., April 1992.
11. Floréen, P., Orponen, P. On the computational complexity of analyzing Hopfield nets. *Complex Systems 3* (1989), 577–587.
12. Fogelman, F., Goles, E., Weisbuch, G. Transient length in sequential iterations of threshold functions. *Discr. Appl. Math. 6* (1983), 95–98.
13. Fogelman, F., Robert, Y., Tchuente, M. *Automata Networks in Computer Science: Theory and Applications.* Manchester University Press, 1987.
14. Franklin, S., Garzon, M. Global dynamics in neural networks. *Complex Systems 3* (1989), 29–36.
15. Franklin, S., Garzon, M. Neural computability. In: *Progress in Neural Networks 1* (ed. O. M. Omidvar). Ablex, Norwood, NJ, 1990. Pp. 128–144.
16. Funahashi, K.-I. On the approximate realization of continuous mappings by neural networks. *Neural Networks 2* (1989), 183–192.
17. Furst, M., Saxe, J. B., Sipser, M. Parity, circuits, and the polynomial-time hierarchy. *Math. Systems Theory 17* (1984), 13–27.
18. Garzon, M., Franklin, S. Global dynamics in neural nets II. Report 89-9, Memphis State Univ., Dept. of Mathematical Sciences, 1989.
19. Garzon, M., Franklin, S. Neural computability II. In: *Proc. of the 3rd Internat. Joint Conf. on Neural Networks, Vol. 1.* IEEE, New York, 1989. Pp. 631-637.
20. Godbeer, G. H., Lipscomb, J., Luby, M. On the Computational Complexity of Finding Stable State Vectors in Connectionist Models (Hopfield Nets). Technical Report 208/88, Dept. of Computer Science, Univ. of Toronto, March 1988.
21. Goldmann, M., Håstad, J., Razborov, A. Majority gates vs. general weighted threshold gates. In: *Proc. of the 7th Ann. Conf. on Structure in Complexity Theory.* IEEE, New York, 1992.
22. Goles, E., Fogelman, F., Pellegrin, D. Decreasing energy functions as a tool for studying threshold networks. *Disrc. Appl. Math. 12* (1985), 261–277.
23. Goles, E., Martínez, S. Exponential transient classes of symmetric neural networks for synchronous and sequential updating. *Complex Systems 3* (1989), 589–597.
24. Goles, E., Martínez, S. *Neural and Automata Networks.* Kluwer Academic, Dordrecht, 1990.
25. Goles, E., Olivos, J. The convergence of symmetric threshold automata. *Info. and Control 51* (1981), 98–104.
26. Hajnal, A., Maass, W., Pudlák, P., Szegedy, M., Turán, G. Threshold circuits of bounded depth. In: *Proc. of the 28th Ann. IEEE Symp. on Foundations of Computer Science.* IEEE, New York, 1987. Pp. 99–110. (Revised version to appear in *J. Comp. Syst. Sci.*)
27. Haken, A. Connectionist networks that need exponential time to stabilize. Manuscript, 10 pp., January 1989.
28. Haken, A., Luby, M. Steepest descent can take exponential time for symmetric connection networks. *Complex Systems 2* (1988), 191–196.
29. Hartley, R., Szu, H. A comparison of the computational power of neural networks. In: *Proc. of the 1987 Internat. Conf. on Neural Networks, Vol. 3.* IEEE, New York, 1987. Pp. 15–22.
30. Håstad, J. Almost optimal lower bounds for small depth circuits. In: *Randomness and Computation. Advances in Computing Research 5* (ed. S. Micali). JAI Press, Greenwich, CT, 1989. Pp. 143–170.
31. Håstad, J. On the size of weights for threshold gates. Manuscript, 11 pp., August 1992.

32. Håstad, J., Goldmann, M. On the power of small-depth threshold circuits. *Computational Complexity 1* (1991), 113–129.
33. Hertz, J., Krogh, A., Palmer, R. G. *Introduction to the Theory of Neural Computation.* Addison-Wesley, Redwood City, CA, 1991.
34. Hinton, G. E., Sejnowski, T. E. Learning and relearning in Boltzmann machines. In [67], pp. 282–317.
35. Hong, J. On connectionist models. *Comm. Pure and Applied Math. XLI* (1988), 1039–1050.
36. Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proc. Nat. Acad. Sci. USA 79* (1982), 2554–2558.
37. Hornik, K., Stinchcombe, M., White, H. Multilayer feedforward nets are universal approximators. *Neural Networks 2* (1989), 359–366.
38. Judd, J. S. On the complexity of loading shallow neural networks. *J. Complexity 4* (1988), 177–192.
39. Judd, J. S. *Neural Network Design and the Complexity of Learning.* The MIT Press, Cambridge, MA, 1990.
40. Kamp, Y., Hasler, M. *Recursive Neural Networks for Associative Memory.* John Wiley & Sons, Chichester, 1990.
41. Karp, R. M., Lipton, R. J. Turing machines that take advice. *L'Enseignement Mathématique 28* (1982), 191–209.
42. Kearns, M., Valiant, L. G. Cryptographic limitations on learning Boolean formulae and finite automata. In: *Proc. of the 21st Ann. ACM Symp. on Theory of Computing.* ACM, New York, 1989. Pp. 433–444.
43. Kleene, S. C. Representation of events in nerve nets and finite automata. In: *Automata Studies* (ed. C. E. Shannon and J. McCarthy). Annals of Mathematics Studies n:o 34. Princeton Univ. Press, Princeton, NJ, 1956. Pp. 3–41.
44. Kohonen, T. *Self-Organization and Associative Memory.* 3rd Ed., Springer-Verlag, Berlin, 1989.
45. Lepley, M., Miller, G. Computational power for networks of threshold devices in an asynchronous environment. Unpublished manuscript, Dept. of Mathematics, Massachusetts Inst. of Technology, 1983.
46. Lin, J.-H., Vitter, J. S. Complexity results on learning by neural nets. *Machine Learning 6* (1991), 211–230.
47. Maass, W., Schnitger, G., Sontag, E. D. On the computational power of sigmoid versus Boolean threshold circuits. In: *Proc. of the 32nd Ann. IEEE Symp. on Foundations of Computer Science.* IEEE, New York, 1991. Pp. 767–776.
48. McClelland, J. L., Rumelhart, D. E., et al. (eds.) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 2.* The MIT Press, Cambridge, MA, 1986.
49. McCulloch, W. S., Pitts, W. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys. 5* (1943), 115–133. Reprinted in [4], pp. 18–27.
50. Minsky, M. L. *Computation: Finite and Infinite Machines.* Prentice-Hall, Englewood Cliffs, NJ, 1972.
51. Minsky, M. L., Papert, S. A. *Perceptrons: An Introduction to Computational Geometry.* The MIT Press, Cambridge, MA, 1969 (expanded edition 1988).
52. Muroga, S. *Threshold Logic and Its Applications.* John Wiley & Sons, New York, 1971.
53. Muroga, S., Toda, I., Takasu, S. Theory of majority decision elements. *J. Franklin Inst. 271* (1961), 376–418.
54. Myhill, J., Kautz, W. H. On the size of weights required for linear-input switching functions. *IRE Trans. Electronic Computers 10* (1961), 288–290.

55. Obradovic, Z., Parberry, I. Analog neural networks of limited precision I: Computing with multilinear threshold functions (Preliminary version). In: *Advances in Neural Information Processing Systems 2* (ed. D. S. Touretzky). Morgan Kaufmann, San Mateo, CA, 1990. Pp. 702–709.
56. Orponen, P. On the computational power of discrete Hopfield nets. Manuscript submitted for publication, 11 pp., November 1992.
57. Parberry, I. 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, 1990. Pp. 217–268.
58. Parberry, I. Circuit Complexity and Neural Networks. Report CRDPC-91-9, Center for Research in Parallel and Distributed Computing. Univ. of North Texas, Denton, TX, 1991. 24 pp.
59. Parberry, I., Schnitger, G. Relating Boltzmann machines to conventional models of computation. *Neural Networks 2 (1989)*, 59–67.
60. Pollack, J. On Connectionist Models of Natural Language Processing. Ph. D. Thesis, Univ. Illinois, Urbana, 1987.
61. Porat, S. Stability and looping in connectionist models with asymmetric weights. *Biol. Cybern. 60* (1989), 335–344.
62. Raghavan, P. Learning in threshold networks. In: *Proc. of the 1988 Workshop on Computational Learning Theory* (ed. D. Haussler, L. Pitt). Morgan Kaufmann, San Mateo, CA, 1988. Pp. 19–27.
63. Reif, J. H., Tate, S. R. On threshold circuits and polynomial computation. *SIAM J. Comput. 21* (1992), 896–908.
64. F. Rosenblatt. *Priciples of Neurodynamics.* Spartan Books, New York, 1962.
65. Roychowdhury, V., Siu, K. Y., Orlitsky, A., Kailath, T. A geometric approach to threshold circuit complexity. In: *Proc. of the 4th Ann. Workshop on Computational Learning Theory* (ed. L. G. Valiant, M. K. Warmuth). Morgan Kaufmann, San Mateo, CA, 1991. Pp. 97–111.
66. Rumelhart, D. E., Hinton, G. E., Williams, R. J. Learning internal representations by error propagation. In [67], pp. 318–362.
67. Rumelhart, D. E., McClelland, J. L., et al. (eds.) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1.* The MIT Press, Cambridge, MA, 1986.
68. Siegelman, H. T., Sontag, E. D. On the computational power of neural nets. In: *Proc. of the 5th Ann. ACM Workshop on Computational Learning Theory.* ACM Press, New York, NY, 1992. Pp. 440–449.
69. Siu, K.-Y., Bruck, J. Neural computation of arithmetic functions. *Proc. of the IEEE 78* (1990), 1669–1675.
70. Siu, K.-Y., Bruck, J. On the power of threshold circuits with small weights. *SIAM J. Discr. Math. 4* (1991), 423–435.
71. Wegener, I. *The Complexity of Boolean Functions.* John Wiley & Sons, Chichester, and B. G. Teubner, Stuttgart, 1987.
72. Wiedermann, J. Complexity issues in discrete neurocomputing. In: *Aspects and Prospects of Theoretical Computer Science. Proc. of the 6th Meeting of Young Computer Scientists* (ed. J. Dassow, J. Kelemen). Lecture Notes in Computer Science 464, Springer-Verlag Berlin Heidelberg 1990. Pp. 93–108.
73. Yao, A. C. Separating the polynomial-time hierarchy by oracles. In: *Proc. of the 26th Ann. IEEE Symp. on Foundations of Computer Science.* IEEE, New York, 1985. Pp. 1–10.

This article was processed using the LaTeX macro package with LLNCS style