Nordic Journal of Computing (to appear)

COMPUTATIONAL COMPLEXITY OF NEURAL NETWORKS: A SURVEY

PEKKA ORPONEN*
Department of Computer Science
P. O. Box 26
FIN-00014 University of 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. Our main emphasis is on the computational power of various acyclic and cyclic network models, but we also discuss briefly the complexity aspects of synthesizing networks from examples of their behavior.

CR Classification: F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*neural networks, circuits*; F.1.3 [**Computation by Abstract Devices**]: Complexity Classes—*complexity hierarchies*

Key words: Neural networks, computational complexity, threshold circuits, associative memory

1. Introduction

The currently again very active field of computation by "neural" networks has opened up a wealth of fascinating research topics in the computational complexity analysis of the models considered. 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 [65].

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 article [72]. which also gives the proofs of

* A preliminary version of this paper appears in *Proc. of the 17th International Symp. on Mathematical Foundations of Computer Science (Prag, Aug. 1992),* Lecture Notes in Computer Science 629, Springer-Verlag, Berlin, 1992, pp. 50–61.

Received December 1995.

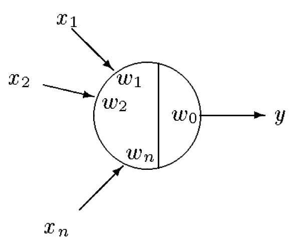

Fig. 1: A formal neuron.

many of the most significant (semi-)elementary results. Our emphasis is at some points slightly different from Parberry's, and we also update his article with some of the more recent developments. Other survey articles of related topics, partially overlapping the present one, are [73, 96]. (At least two more comprehensive books [74, 75] are also in preparation.) As a general introduction to neural networks theory, although mostly other than complexity aspects, the excellent textbook [40] can be recommended, and certain aspects of computations in cyclic networks are covered in depth in [48]. The original "PDP" books [86, 61] still make very inspiring reading, and many significant original papers have been reprinted in the anthologies [6, 7]. Also, earlier texts on threshold logic such as the comprehensive [66] 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.

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 [18, 30]).

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, [70]), or often just to integers from $\{-1, 1\}$, $\{0, 1\}$, or $\{-1, 0, 1\}$.

The transfer function σ may be either a discrete step function:

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

or a smooth "sigmoid" such as

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

or possibly a piecewise linear or polynomial 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 [41, 1], where its behavior moreover depends on a time-dependent "temperature" parameter T:

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

Neurons with a step transfer function are commonly called *perceptrons* (a name introduced by Rosenblatt in his seminal work [83]), or *threshold gates* in the case of binary valued inputs. An important special type of threshold gate is the *majority gate*, where all the weights equal 1, except the bias term, which equals half the number of input lines.

A binary valued function defined on some set of points in the input space and computable by a single perceptron is called *linearly separable.* The name derives from the fact that a perceptron basically implements a hyperplane in the input space, dividing the points with output 1 from the points with output 0. A linearly separable function defined on the corners of a binary hypercube (i.e., a Boolean function computable by a single threshold gate) is called a *threshold function.*

A neural 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 updates may be *continuous*, in which case the behavior of the network is described by a set of differential equations, or they may be *discrete*, in which case the differential equations are replaced by iterated functions. We shall here consider only discrete-time models, although the continuous ones are also of considerable importance (for references, see [40]).

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 $\vec{x}$ of a cyclic network is *stable* if none of the neurons would change its state in an update. (In matrix notation, $\sigma(W\vec{x} - \vec{w}_0) = \vec{x}$, where W is the connection weight matrix and $\vec{w}_0$ is the threshold vector.) 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 these and other models, see [40, 53, 86]):

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 step 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 *majority circuit* where all the neurons are simple majority gates. An important intermediate model is the *threshold circuit* or “multilayer perceptron”, where the weights may be arbitrary reals, but the input and output signals are discretized to binary values by step 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 the simple majority 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 ([67, 66]; for recent versions of the proof, see [37, 42, 72, 73, 80]):

THEOREM 1. *Any threshold function on n variables can be computed by a threshold gate 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 [37] (although weaker versions of the converse have been known since the early 60's, cf. [66, 68, 72]):

THEOREM 2. *For infinitely many n, there are threshold functions on n variables whose computation by a single threshold gate requires weights as large as $n^{n/2}/2^n$.*

These theorems imply that polynomial size threshold 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 threshold circuit with small weights can be easily implemented as a polynomial size majority circuit of the same depth [72], and a polynomial size threshold circuit with large weights can be implemented as a polynomial size majority circuit of at most twice the depth [26].

In analogy to the standard AND/OR/NOT circuit complexity classes AC^k and NC^k [95], the following threshold, or equivalently majority circuit complexity classes are defined for each $k \geq 0$:

$$\begin{aligned} \mathrm{TC}^k \;=\; & \{\text{functions computable by threshold circuits of} \\ & \text{polynomial size and depth } O(\log^k n)\}. \end{aligned}$$

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 [22, 98, 36, 95], but can of course be computed by a single majority 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 majority circuits are in fact 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 [66, 72], the product of two n-bit numbers can be computed in depth 3 [92], and analytic functions with a convergent rational power series (e.g., sin, cos, exp, log, sqrt) can all be approximated in bounded depth [82] (cf. also [90]). It is therefore of great interest to consider also the following fixed-depth sublevels of the class TC^0 individually:

$$\begin{aligned}\mathrm{TC}^0_d &= \{\text{functions computable by threshold circuits of}\\ &\qquad \text{polynomial size and depth } d\},\\ \widehat{\mathrm{TC}}^0_d &= \{\text{functions computable by majority circuits of}\\ &\qquad \text{polynomial size and depth } d\}.\end{aligned}$$

(Some authors, e.g. [26, 91], use the notations LT_d, $\widehat{\mathrm{LT}}_d$, from "linear thresholds", for these classes.) It is shown in [26] that the classes $\widehat{\mathrm{TC}}^0_d$, TC^0_d form a hierarchy:

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

for all $d \geq 1$. (For a constructive version of the proof, see [27].) Separating the levels of this hierarchy, as far as they are separate, is currently a significant research task.

That class $\widehat{\mathrm{TC}}^0_1$ is properly contained in TC^0_1 follows from Theorem 2 above; and the proper containment of TC^0_1 in $\widehat{\mathrm{TC}}^0_2$ follows from the well-known fact [65] that the parity function cannot be computed by a single threshold gate, but being a symmetric function, it can be computed by a small majority circuit of depth 2. The classes $\widehat{\mathrm{TC}}^0_2$ and TC^0_2 are separated by a result in [26]. The separation of $\widehat{\mathrm{TC}}^0_2$ from $\widehat{\mathrm{TC}}^0_3$ was first proved in [32]. 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 [32] basically consists in showing that the ip_2 function correlates only weakly with any function computable by a threshold gate with polynomially bounded weights, but strong correlation would be necessary for the function to be computable by a small two-layer circuit of majority gates. (However, the function could still conceivably be computable by a small two-layer circuit of unbounded-weight threshold gates.) This important technique has recently been simplified and used to prove new results in [26, 38, 84].

The separation of $\widehat{\mathrm{TC}}^0_2$ from $\widehat{\mathrm{TC}}^0_3$ stands in contrast to the often-cited results in the neural networks literature [12, 21, 45] proving that two-layer backpropagation nets with sigmoid transfer functions "suffice", 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 this representation is exponentially large in the number of inputs [95].

Still, there could in principle be some representational efficiency to be gained from using continuous instead of step transfer functions. This question was studied in [60], 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 $d \geq 1$,

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

where $\widehat{\mathrm{TC}}_d^0(\sigma)$ 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 [76, 72], to the basic deterministic threshold 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 (i.e., not describable by an effective algorithm).

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 convergence procedure" [83, 65, 40] for single perceptrons, and the more recent "backpropagation algorithm" [85] 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 [46, 47], who proved that the following version is NP-complete:

Threshold Circuit Loading. Given a set of pairs $\{(\vec{x}_1, b_1), \ldots, (\vec{x}_m, b_m)\}$, where each $\vec{x}_i \in \{0,1\}^n$ 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 threshold circuit, $f(\vec{x}_i) = b_i$ 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 [10], 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

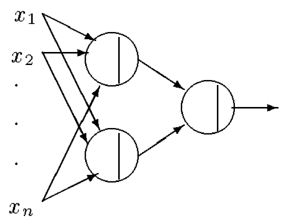

Fig. 2: A three-node network.

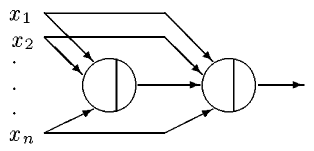

Fig. 3: A "cascade" two-node network.

are connected. Lin and Vitter [58] 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:

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

It should be pointed out that the problem of whether a given set of input/output pairs can be implemented by a *single* threshold gate can be solved in polynomial time by linear programming techniques. (The requirements for correct loading can be written as a system of linear inequalities, with the weights as unknowns. This system can then be solved by, e.g., some variant of the ellipsoid method [88].) 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 gate 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 [50] 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.

Interestingly, despite all these results on the synthesis of threshold circuits, the complexity of the learning problem in backpropagation nets with continuous transfer functions still seems to be an open question.

4. Cyclic nets

4.1 Computational power

It has been known since the early work of McCulloch and Pitts [62] and Kleene [51] (see also [64]) 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 [20, 35], and constructions using a finite network, but real-valued neurons of arbitrary precision are presented in [35, 78, 89]. The construction in [89] is rather interesting, as there the required precision grows only linearly in the space requirement of the simulated Turing machine, and only very simple "saturated linear" transfer functions are used.

In the McCulloch–Pitts–Kleene model and its extensions, the input is presented as a sequence of pulses to a net that is capable of processing arbitrary long input sequences. 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) converges, 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 formulation naturally requires that the network grow with increasing input size, i.e., that we actually consider nonuniform *sequences* of networks, one for each input size. If the only part of the network that changes is the set of input neurons, then this model is computationally equivalent to the sequential input, uniform network model. However, if also changes in the network structure are possible, then the situation is fundamentally different.

Since the computations of a cyclic net of size s converging 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 [8, 49]. (The construction is actually quite simple: one starts with the standard simulation of polynomial space bounded Turing machines by acyclic nets of polynomial width and exponential depth [9]. A moment of thought shows that all the layers of gates in the resulting net are in fact similar, and so the net can be "folded" upon itself to create a cyclic net of polynomial size and exponential convergence time. Parberry [72] attributes

[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 $\Omega((n \log n)^{1/3})$ gates in the worst case.

this result to the unpublished report [57].) For approaches to computations on nets of unbounded size, see [19, 20, 23, 24].

Concerning symmetric nets, the fundamental result is Hopfield's [43] observation that a symmetric simple net using an asynchronous update rule always converges. Hopfield's result was based on defining an "energy" function on the global states of the network, such that in every permissible update of a neuron state, the energy decreases. By a more careful consideration of the energy function, the result was improved in [17] to show that in a symmetric simple network of n neurons with integer weights w_{ij}, the convergence 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 any asynchronous update rule. Under synchronous updates, a similar time bound holds [28], but the network may also converge to oscillate between two alternating states instead of a unique stable state (see also [11]). Thus, in particular, 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 in [31] for synchronous updates (a simplified construction appears in [29]), and in [34] for a particular asynchronous update rule. (For a different update rule the latter result actually follows already from [31] or [29] by a fairly simple construction.) Finally, a network requiring exponential time for convergence under any asynchronous update rule was demonstrated in [33]. This result is now also known to follow from the more general theory of "PLS-completeness" for local optimization problems [87]. Related work appears in [11, 30].

Somewhat surprisingly, the very constrained convergence behavior of symmetric nets is not reflected in their computational power, at least not when under synchronous updates. 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 [71].

4.2 Synthesis

The most significant application areas for cyclic neural networks proposed so far are associative (or error-correcting) memory, and solving combinatorial optimization problems. For optimization applications, Hopfield and Tank [44] suggested mapping a given instance of, say, a minimization problem to a symmetric network whose energy function encodes the relevant costs and constraints, and then letting the network solve the problem by converging to a (local) minimum of its energy. The idea is intriguing, but so far the results have been mixed (cf. e.g. [97]). We shall not consider this application area further here.

For associative memory, the idea is to construct from a given set of binary vectors $\vec{x}_1, \ldots, \vec{x}_m$ a cyclic network that has at least these vectors as its stable global states, preferably so that when the network is initialized in some state that is close to one of the vectors, $\vec{x}_i$, it will quickly correct itself and converge to $\vec{x}_i$. The set of all vectors that are guaranteed to eventually converge to a given $\vec{x}_i$ forms the *attraction domain* of $\vec{x}_i$. The *attraction radius* of $\vec{x}_i$ is the largest Hamming distance from within which all other vectors are guaranteed eventually to converge to $\vec{x}_i$[2].

A *storage rule* is an algorithm for constructing, from a given set of vectors $\vec{x}_i$, a cyclic net that stores those vectors, or at least a net that with high probability stores most of the $\vec{x}_i$ approximately correctly. The *capacity* of a storage rule is, roughly speaking, the maximum number of random vectors it can store with high probability, as a function of the number of neurons n in the network. This notion can be made precise in several alternative ways, depending on whether all the given vectors must be stored, whether they need to be stored exactly, what is required of the attraction radii, etc.

Several different storage rules have been proposed and analyzed in the literature. The two most prominent varieties are the *outer product* rule popularized by Hopfield in [43] (but actually studied already in the 70's by Anderson [5], Kohonen [52], Nakano [69], and others), and the various *spectral* or *pseudoinverse* algorithms discussed recently in [14, 77, 94] (and in the 70's in at least [54]).

In the outer product rule, the weight matrix W is derived from the vectors $\vec{x}_i$, which we may for simplicity assume to have components $-1/1$, as $W = \sum_{i=1}^{m}(\vec{x}_i(\vec{x}_i)^T - I)$. The threshold vector is set to zero. The rule reflects a form of "Hebbian learning" [39]: two similarly active components in the stored vectors tend to create an excitatory (positive) interconnection between the respective neurons, whereas opposing activities in the components tend to create inhibitory (negative) connections.

The capacities achieved by the outer product rule are not too good. With high probability, only $O(n/\log n)$ random n-bit vectors can be stored so that all, or even most of them are recalled correctly [56, 63, 94]. If a limited number of errors can be tolerated in the recall of the stored vectors, then the probabilistic capacity reaches $0.138n$ [4, 43].

Better performance can be obtained from the spectral algorithms, where the goal is to construct the weight matrix W so that the vectors $\vec{x}_i$, if they are linearly independent, become its eigenvectors. This can be achieved by choosing, e.g., $W = X(X^TX)^{-1}X^T$, where X is a column matrix of the vectors $\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_m$. This method obviously guarantees perfect retrieval of any set of linearly independent vectors. Since the probability that n random binary vectors are linearly independent approaches 1 as $n \to \infty$ [55], the probabilistic capacity of the spectral algorithms is n.

[2] Note that while the set of stable states of a network does not depend on the update rule, a given unstable state can in sequential operation converge to different stable states, depending on the update order. In the definition of attraction domains and radii for asynchronous nets, convergence should be required *irrespective* of the update order chosen.

In general, the problem of deciding whether a given set of vectors can be stored as stable states of a Hopfield network can again be solved in polynomial time by linear programming, as the constraints on the interconnection weights can be expressed as a system of linear inequalities. Of course, this technique gives no control on the attraction radii of the stable states — in fact, the complexity of the following version of the problem is apparently not known:

Hopfield Net Loading. Given a set of vectors $\{\vec{x}_1, \ldots, \vec{x}_m\}$, where each $\vec{x}_i \in \{-1, 1\}^n$, and a constant ρ, is there a symmetric net of n neurons that has these vectors as stable states, with attraction radii $\geq \rho$?

It seems to be not known even whether the problem is in P or NP-hard for any fixed $\rho \geq 1$. Of course, all the storage rules proposed in the literature provide in some sense approximate solutions to this problem.

While the complexity of synthesizing a cyclic net from a given set of patterns is not known, several results point to the difficulty of *analyzing* a ready-made cyclic net. For instance:

(1) It is an NP-hard problem to decide if a given asymmetric network under synchronous updates will converge from all initial states [79], or from any initial state [2, 25]. (Actually, it follows from the network construction in [71] that these problems are PSPACE-complete.)

(2) It is an NP-complete problem to decide if a given symmetric, simple network has more than one stable state [25]. In fact, the problem of counting the number of stable states is #P-complete [16]. (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) [16]. 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. In fact, the attraction radius cannot be even approximated to within a factor of $n^{1-\epsilon}$ for any fixed $\epsilon > 0$, unless P = NP [15].

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 [98, 36]? All the separations from TC_2^0 upwards are open. (Partial results appear in [38, 81].) 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 [60] on the equivalence of sigmoid and step transfer functions for small-weight networks to arbitrary weights; i.e., to prove that for any "reasonable" nonlinear transfer function σ,

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

holds for all $d \geq 1$. (Partial results in this direction are presented in [13, 59]. However, simple smoothness conditions are not sufficient to constrain the function σ here, as in [93] an arbitrarily smooth σ is constructed that is capable of representing *any* Boolean function in a network of just two neurons with sufficiently large weights.)

For cyclic nets, it would be most interesting to know the exact complexity of the Hopfield net loading problem mentioned in section 4.2. A number of interesting issues remain to be explored also in determining the computational power of nets under asynchronous and continous-time updates.

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] Amit, D. J., Gutfreund, H., Sompolinsky, H. Storing infinite numbers of patterns in a spin-glass model of neural networks. *Physical Review Letters 55* (1985), 1530–1533.

[5] Anderson, J. A. A simple neural network generating an interactive memory. *Mathematical Biosciences 14* (1972), 197–220. Reprinted in [6], pp. 181–192.

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

[7] Anderson, J. A., Pellionisz, A., Rosenfeld, E. (eds.) *Neurocomputing 2: Directions for Research.* The MIT Press, Cambridge, MA, 1990.

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

[9] Balcázar, J. L., Díaz, J., Gabarró, J. *Structural Complexity I.* Springer-Verlag, Berlin, 1988.

[10] Blum, A. L., Rivest, R. L. Training a 3-node neural network in NP-complete. *Neural Networks 5* (1992), 117–127.

[11] Bruck, J. On the convergence properties of the Hopfield model. *Proc. of the IEEE 78* (1990), 1579–1585.

[12] Cybenko, G. Approximation by superposition of a sigmoidal function. *Math. of Control, Signals, and Systems 2* (1989), 303–314.

[13] DasGupta, B., Schnitger, G. The power of approximating: A comparison of activation functions. In: *Advances in Neural Information Processing Systems 5* (ed. S. J. Hanson, J. D. Cowan, C. L. Giles). Morgan Kaufmann, San Mateo, CA, 1993. Pp. 615–622.

[14] Dembo, A. On the capacity of associative memories with linear threshold functions. *IEEE Trans. on Information Theory 35* (1989), 709–720.

[15] Floréen, P., Orponen, P. Attraction radii in binary Hopfield nets are hard to compute. *Neural Computation 5* (1993), 812–821.

[16] Floréen, P., Orponen, P. On the computational complexity of analyzing Hopfield nets. *Complex Systems 3* (1989), 577–587.

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

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

[19] Franklin, S., Garzon, M. Global dynamics in neural networks. *Complex Systems 3* (1989), 29–36.

[20] Franklin, S., Garzon, M. Neural computability. In: *Progress in Neural Networks 1* (ed. O. M. Omidvar). Ablex, Norwood, NJ, 1990. Pp. 128–144.

[21] Funahashi, K.-I. On the approximate realization of continuous mappings by neural networks. *Neural Networks 2* (1989), 183–192.

[22] Furst, M., Saxe, J. B., Sipser, M. Parity, circuits, and the polynomial-time hierarchy. *Math. Systems Theory 17* (1984), 13–27.

[23] Garzon, M., Franklin, S. Global dynamics in neural nets II. Report 89-9, Memphis State Univ., Dept. of Mathematical Sciences, 1989.

[24] 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.

[25] 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.

[26] Goldmann, M., Håstad, J., Razborov, A. Majority gates vs. general weighted threshold gates. *Computational Complexity 2* (1992), 277–300.

[27] Goldmann, M., Karpinski, M. Simulating threshold circuits by majority circuits. In: *Proc. of the 25th Ann. ACM Symp. on Theory of Computing.* ACM, New York, 1993. Pp. 551–560.

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

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

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

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

[32] Hajnal, A., Maass, W., Pudlák, P., Szegedy, M., Turán, G. Threshold circuits of bounded depth. *J.Comput. Syst. Sci. 46* (1993), 129–154. (Preliminary version in: *Proc. of the 28th Ann. IEEE Symp. on Foundations of Computer Science.* IEEE, New York, 1987. Pp. 99–110.)

[33] Haken, A. Connectionist networks that need exponential time to stabilize. Unpublished manuscript, Dept. of Computer Science, University of Toronto, 1989.

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

[35] 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.

[36] 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.

[37] Håstad, J. On the size of weights for threshold gates. *SIAM J. Discr. Math.*, to appear.

[38] Håstad, J., Goldmann, M. On the power of small-depth threshold circuits. *Computational Complexity 1* (1991), 113–129.

[39] Hebb, D. O. *The Organization of Behavior.* John Wiley & Sons, New York, NY, 1949.
[40] Hertz, J., Krogh, A., Palmer, R. G. *Introduction to the Theory of Neural Computation.* Addison-Wesley, Redwood City, CA, 1991.
[41] Hinton, G. E., Sejnowski, T. E. Learning and relearning in Boltzmann machines. In [86], pp. 282–317.
[42] Hong, J. On connectionist models. *Comm. Pure and Applied Math. XLI* (1988), 1039–1050.
[43] Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proc. Nat. Acad. Sci. USA 79* (1982), 2554–2558.
[44] Hopfield, J. J., Tank, D. W. Neural computation of decisions in optimization problems. *Biological Cybernetics 52* (1985), 141–152.
[45] Hornik, K., Stinchcombe, M., White, H. Multilayer feedforward nets are universal approximators. *Neural Networks 2* (1989), 359–366.
[46] Judd, J. S. On the complexity of loading shallow neural networks. *J. Complexity 4* (1988), 177–192.
[47] Judd, J. S. *Neural Network Design and the Complexity of Learning.* The MIT Press, Cambridge, MA, 1990.
[48] Kamp, Y., Hasler, M. *Recursive Neural Networks for Associative Memory.* John Wiley & Sons, Chichester, 1990.
[49] Karp, R. M., Lipton, R. J. Turing machines that take advice. *L'Enseignement Mathématique 28* (1982), 191–209. (Preliminary version in: *Proc. of the 12th Ann. ACM Symp. on Theory of Computing.* ACM, New York, 1980. Pp. 302–309.)
[50] Kearns, M., Valiant, L. G. Cryptographic limitations on learning Boolean formulae and finite automata. *J. Assoc. Comput. Mach. 41* (1994), 67–95. (Preliminary version in: *Proc. of the 21st Ann. ACM Symp. on Theory of Computing.* ACM, New York, 1989. Pp. 433–444.)
[51] 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.
[52] Kohonen, T. Correlation matrix memories. *IEEE Trans. on Computers 21* (1972), 353–359. Reprinted in [6], pp. 174–180.
[53] Kohonen, T. *Self-Organization and Associative Memory.* 3rd Ed., Springer-Verlag, Berlin, 1989.
[54] Kohonen, T., Ruohonen, K. Representation of associated data by matrix operations. *IEEE Trans. on Computers 22* (1973), 701–702.
[55] Komlós, J. On the determinant of (0,1) matrices. *Stud. Sci. Math. Hungarica 2* (1967), 7–21.
[56] Kuh, A., Dickinson, B. W. Information capacity of associative memories. *IEEE Trans. on Information Theory 35* (1989), 59–68.
[57] 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.
[58] Lin, J.-H., Vitter, J. S. Complexity results on learning by neural nets. *Machine Learning 6* (1991), 211–230.
[59] Maass, W. Bounds for the computational power and learning complexity of analog neural nets. In: *Proc. of the 25th Ann. ACM Symp. on Theory of Computing.* ACM, New York, 1993. Pp. 335–344.
[60] 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.
[61] 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.
[62] McCulloch, W. S., Pitts, W. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys. 5* (1943), 115–133. Reprinted in [6], pp. 18–27.

[63] McEliece, R. J., Posner, E. C., Rodemich, E. R., Venkatesh, S. S. The capacity of the Hopfield associative memory. *IEEE Trans. on Information Theory 33* (1987), 461–482.

[64] Minsky, M. L. *Computation: Finite and Infinite Machines.* Prentice-Hall, Englewood Cliffs, NJ, 1967.

[65] Minsky, M. L., Papert, S. A. *Perceptrons: An Introduction to Computational Geometry.* The MIT Press, Cambridge, MA, 1969 (expanded edition 1988).

[66] Muroga, S. *Threshold Logic and Its Applications.* John Wiley & Sons, New York, 1971.

[67] Muroga, S., Toda, I., Takasu, S. Theory of majority decision elements. *J. Franklin Inst. 271* (1961), 376–418.

[68] Myhill, J., Kautz, W. H. On the size of weights required for linear-input switching functions. *IRE Trans. Electronic Computers 10* (1961), 288–290.

[69] Nakano, K. Associatron — a model of associative memory. *IEEE Trans. on Systems, Man, and Cybernetics 12* (1972), 380–388. Reprinted in [7], pp. 90–98.

[70] 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.

[71] Orponen, P. On the computational power of discrete Hopfield nets. In: *Proc. 20th International Colloquium on Automata, Languages, and Programming* (eds. S. Carlsson, R. Karlsson, A. Lingas). Lecture Notes in Computer Science 700, Springer-Verlag, Berlin, 1993. Pp. 215–226.

[72] 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.

[73] 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.

[74] Parberry, I. *Circuit Complexity and Neural Networks.* The MIT Press, Cambridge, MA, to appear in 1994.

[75] Parberry, I. (ed.) *The Computational and Learning Complexity of Neural Networks: Advanced Topics.* In preparation.

[76] Parberry, I., Schnitger, G. Relating Boltzmann machines to conventional models of computation. *Neural Networks 2 (1989)*, 59–67.

[77] Personnaz, L., Guyon, I., Dreyfus, G. Collective computational properties of neural networks: New learning mechanisms. *Physical Review A 34* (1986), 4217–4228.

[78] Pollack, J. On Connectionist Models of Natural Language Processing. Ph. D. Thesis, Univ. Illinois, Urbana, 1987.

[79] Porat, S. Stability and looping in connectionist models with asymmetric weights. *Biol. Cybern. 60* (1989), 335–344.

[80] 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.

[81] Razborov, A., Wigderson, A. $n^{\Omega(\log n)}$ lower bounds on the size of depth-3 threshold circuits with AND gates at the bottom. *Info. Proc. Letters 45* (1993), 303–307.

[82] Reif, J. H., Tate, S. R. On threshold circuits and polynomial computation. *SIAM J. Comput. 21* (1992), 896–908.

[83] F. Rosenblatt. *Priciples of Neurodynamics.* Spartan Books, New York, 1962.

[84] 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.

[85] Rumelhart, D. E., Hinton, G. E., Williams, R. J. Learning internal representations by error propagation. In [86], pp. 318–362.

[86] 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.

[87] Schäffer, A. A., Yannakakis, M. Simple local search problems that are hard to solve. *SIAM J. Comput. 20* (1991), 56–87.

[88] Schrijver, A. *Theory of Linear and Integer Programming.* John Wiley & Sons, Chichester, 1986.

[89] Siegelmann, 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.

[90] Siu, K.-Y., Bruck, J. Neural computation of arithmetic functions. *Proc. of the IEEE 78* (1990), 1669–1675.

[91] Siu, K.-Y., Bruck, J. On the power of threshold circuits with small weights. *SIAM J. Discr. Math. 4* (1991), 423–435.

[92] Siu, K.-Y., Roychowdhury, V. Optimal depth neural networks for multiplication and related problems. In: *Advances in Neural Information Processing Systems 5* (ed. S. J. Hanson, J. D. Cowan, C. L. Giles). Morgan Kaufmann, San Mateo, CA, 1993. Pp. 59–64.

[93] Sontag, E. D. Feedforward nets for interpolation and classification. *J. Comput. Syst. Sci. 45* (1992), 20–48.

[94] Venkatesh, S. S., Psaltis, D. Linear and logarithmic capacities in associative neural networks. *IEEE Trans. on Information Theory 35* (1989), 558–568.

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

[96] 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.

[97] Wilson, G. V., Pawley, G. S. On the stability of the travelling salesman problem algorithm of Hopfield and Tank. *Biological Cybernetics 57* (1988), 63–70.

[98] 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.