Complexity Issues in Discrete Hopfield Networks*

Patrik Floréen and Pekka Orponen

University of Helsinki, Department of Computer Science

P. O. Box 26, FIN-00014 University of Helsinki, Finland

Abstract

We survey some aspects of the computational complexity theory of discrete-time and discrete-state Hopfield networks. The emphasis is on topics that are not adequately covered by the existing survey literature, most significantly:

1. *the known upper and lower bounds for the convergence times of Hopfield nets (here we consider mainly worst-case results);*
2. *the power of Hopfield nets as general computing devices (as opposed to their applications to associative memory and optimization);*
3. *the complexity of the synthesis ("learning") and analysis problems related to Hopfield nets as associative memories.*

*Draft chapter for the forthcoming book *The Computational and Learning Complexity of Neural Networks: Advanced Topics* (ed. Ian Parberry).

1 Introduction

A (discrete) *Hopfield network* [41][70, Section 8.3] consists of n binary valued nodes. We index the nodes by $\{1, \ldots, n\}$, and choose $\{-1, +1\}$ as their possible states[1]. The *size* of a network is its number of nodes. Associated to each pair of nodes i, j is a *connection weight* w_{ij}. In addition, each node i has an internal *threshold value* h_i. We denote the matrix of connection weights by $W = (w_{ij})$, and the vector of threshold values by $h = (h_1, h_2, \ldots, h_n)$. In Hopfield networks proper, the connection matrix W is symmetric, so that $w_{ij} = w_{ji}$ for each i, j. In this survey, however, we shall occasionally discuss also the more general class of *asymmetric* networks. A Hopfield network is *simple* if $w_{ii} = 0$ for all i, and *semisimple* if $w_{ii} \geq 0$ for all i. A fundamental fact [65, 39, 74][70, Section 4.2] is that for a network of size n it suffices to consider integer connection weights and thresholds of absolute value at most $(n+1)^{(n+1)/2}$, i.e., of length $O(n \log n)$ bits. We say that a network has *small weights* if the absolute values of the weights are polynomially bounded in n. (Strictly speaking, this notion can only be defined when a *sequence* of networks is considered.) The threshold values can then also be assumed to be polynomially bounded. The *(total) weight* of a network is defined as the sum of the absolute values of its connection weights.

At any time t, each node i in a network has a state x_i, which is either -1 or $+1$. The

[1] Instead of this *bipolar* choice of values the *binary* pair $\{0, 1\}$ could have been chosen equally well. Bipolar states x and binary states y are interchangeable via the correspondence $y = \frac{1}{2}(1 + x)$, $x = 2y - 1$.

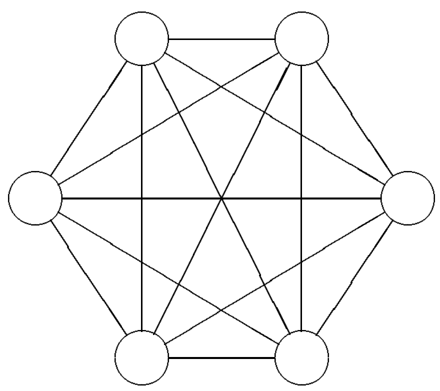

Figure 1: A Hopfield network of size 6.

state at time $t+1$ is determined as a function of the states of the nodes at time t by the formula $x_i^{(t+1)} := \mathrm{sgn}(\sum_{j=1}^{n} w_{ij} x_j^{(t)} - h_i)$, where sgn is the signum function ($\mathrm{sgn}(x) = 1$ for $x \geq 0$ and $\mathrm{sgn}(x) = -1$ for $x < 0$). Node i is *stable* if $x_i = \mathrm{sgn}(\sum_{j=1}^{n} w_{ij} x_j - h_i)$.

In *fully parallel* operation, the update step is performed simultaneously for all the nodes. Thus the global update rule for the network may be denoted as $x := \mathrm{sgn}(Wx - h)$, where $x = (x_1, x_2, \ldots, x_n)$ is the vector of states for the nodes. It is known [72, 28] that, starting from any initial vector of states, a sequence of such updates will eventually converge either to a stable vector (i.e., a vector u such that $\mathrm{sgn}(Wu - h) = u$), or to a cycle of length two (i.e., vectors $u \neq v$ such that $\mathrm{sgn}(Wu - h) = v$ and $\mathrm{sgn}(Wv - h) = u$).

In *sequential* operation, the update step is performed for one node at a time in some (possibly random) order. As shown in [41], in a semisimple network all sequential computations eventually converge to stable vectors (cf. also [70, Section 8.3]). A sequential computation is called *productive* if the state of some node is changed in each update step.

In contrast to the simple convergence behavior of symmetric networks, the behavior of asymmetric networks can be quite complicated and difficult to analyze. We shall return to this issue in Section 3, where we show that it is a $\mathcal{PSPACE}$-complete problem to decide whether an asymmetric network converges to a stable state vector from a given initial state vector.

Two very good books dealing with many aspects of Hopfield networks are those by Kamp and Hasler [46], and by Hertz, Krogh and Palmer [38]. Related network models for associative memory applications are discussed by Kohonen in [53]. For general introductions to computational complexity issues in neural networks, see the book [70], and the survey articles [68, 69, 87].

The most typical applications of Hopfield networks are as associative memories, and for solving combinatorial optimization problems. An *(auto)associative content-addressable memory* is a storage device that stores vectors u^i, for $i = 1, 2, \ldots, m$, in such a way that upon presentation of an input vector x, which in some predefined metric is close to u^i, the memory will give u^i as result (cf. Figure 2). The distance measure we shall use here is the *Hamming distance*, i.e., the number of differing elements in the vectors. Associative memories, being error-correcting devices, have many applications in, e.g., decoding.

The computation by a Hopfield network can be viewed as a search for the minimum of

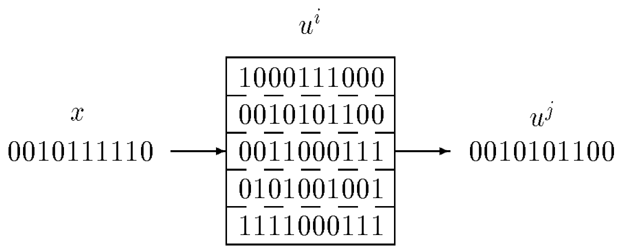

Figure 2: When the vectors u^i are stored, input x, which (according to the given metric) is near some vector u^j, gives as result the vector u^j.

a specific energy function defined on the states of the networks; the local minima of this function correspond to the stable vectors in the network. By encoding a combinatorial minimization problem so that its cost function corresponds to the energy function of a network, local optima of the function can be found by network computation. There exists also a *continuous* version of the Hopfield network [42] which may be better suited for optimization problems. A seminal paper on this topic is [43], where a mapping of the Traveling Salesman Problem to continuous Hopfield nets is described. (However, this method for solving the TSP has subsequently been criticized as too sensitive to the choice of certain parameters affecting the computation [89].)

For associative memory applications, several alternative network synthesis methods or *memory storage rules*, i.e., methods for choosing the connection weights and the threshold values, have been considered. The best-known is the *tensor product* or *outer product* rule [41] where, given vectors $u^1, \ldots, u^m$ to be stored, the weight matrix is chosen as $W = \sum_{i=1}^{m}(u^i(u^i)^{\mathrm{T}} - I)$, and the threshold vector as $h = \overline{0}$. We shall discuss this and other storage rules later on, in Section 4.

As an associative memory the Hopfield network has some shortcomings. Firstly, we can get output vectors that are not stored vectors. These are called *spurious* stable vectors. Secondly, as will be discussed in Section 4.2, the capacity of such a network for reliable storage is quite low, i.e., if too many vectors are loaded onto the network, then some of them cannot be retrieved. In the worst case, a standard Hopfield network with at least one zero on the diagonal of the connection matrix can store with perfect recall only *one* vector, regardless of the storage rule used [12, 64]. However, the Hopfield network is inherently fault tolerant as the network may still give reasonable results even if some nodes are out of operation.

The Hopfield network has been implemented in hardware. The problem in hardware implementations is the vast number of connections. Optical implementations may be a solution to this difficulty. See the collections [44, 75] for papers on hardware implementations of neural networks.

2 Convergence Properties

In this section we derive the currently known upper and lower bounds on the worst-case convergence times of Hopfield networks under sequential and fully parallel updating. Experimental and theoretical work on the expected behavior of the networks is mentioned only briefly.

2.1 Upper Bounds on the Convergence Time

The Hopfield network computation can be viewed as a search for a minimum of a so called *energy function* or *Lyapunov function*[2]. Such a function is a real-valued function of the vector of states, with the property that the value of the function decreases in each update step. Stable vectors of the network correspond to local minima of the energy function. The energy function technique is most useful for proving convergence.

The result below is based on work by Fogelman, Goles et al. [20, 26, 28, 30]. The energy function usually applied is $E(x) = -\frac{1}{2}x^{\mathrm{T}}Wx - h^{\mathrm{T}}x$, but we use instead a slightly modified function introduced in [16]. Our energy function is strictly decreasing in each update step, while the standard one is decreasing, but not strictly decreasing.

Theorem 2.1 *Let e_i be 1 if $\sum_j w_{ij} - h_i$ is even, and 0 otherwise. Any productive sequential computation of a semisimple Hopfield network with integer weights converges to a stable vector in time at most $(\frac{1}{2}\sum_i \sum_{j\neq i} |w_{ij}| + \sum_i |h_i - e_i|)/(1 + \min_k w_{kk})$.*

NOTE: If the network is not semisimple, i.e., if some of the diagonal weights w_{ii} are negative, the network may not converge to a stable vector at all.

PROOF: Choose the energy function $E(x) = -\frac{1}{2}x^{\mathrm{T}}Wx + (h-e)^{\mathrm{T}}x$. Here the vector $e = (e_1, e_2, \ldots, e_n)$, defined as above, ensures that the expression $\sigma_k = \sum_j w_{kj}x_j - h_k + e_k$ is never zero. Assume that in an update step the element k is changed from x_k to x'_k. Then the difference in energy can easily be computed as $\Delta E = (x_k - x'_k)[(\sum_j w_{kj}x_j - h_k + e_k) - \frac{1}{2}w_{kk}(x_k - x'_k)]$. Now if $x_k = -1$, then $x'_k = 1$ and $\sigma_k \geq 1$; thus $\Delta E \leq -2 - 2w_{kk}$; and if $x_k = 1$, then $x'_k = 1$ and $\sigma_k \leq -1$; thus $\Delta E \leq -2 - 2w_{kk}$. As the space $\{-1, 1\}^n$ is discrete, the energy function is bounded, so the computation must eventually stop: a

[2] Sometimes also called “stability” [70] or “harmony” [80].

stable vector is reached. Clearly, $-\frac{1}{2}\sum_i\sum_{j\neq i}|w_{ij}| - \frac{1}{2}\sum_i|w_{ii}| - \sum_i|h_i - e_i| \leq E(x) \leq \frac{1}{2}\sum_i\sum_{j\neq i}|w_{ij}| - \frac{1}{2}\sum_i|w_{ii}| + \sum_i|h_i - e_i|$ for all x. This gives the bound on the convergence time. □

By a more thorough analysis of the maximal energy difference [17], we get the upper bound $(M + \sum_i|h_i - e_i|)/(1 + \min_k w_{kk})$, where M is the sum of the $\lfloor n^2/4 \rfloor$ largest elements in the multiset $\{\,|w_{ij}|\,\big|\, i < j\,\}$. With the tensor product storage rule, we get $|w_{ij}| \leq m$, and hence the bound on the convergence time is $mn^2/4 + n$, where m is the number of stored vectors and n is their length. For real-valued weights and thresholds, the bound is $(\frac{1}{2}\sum_i\sum_{j\neq i}|w_{ij}| + \sum_i|h_i|)/\epsilon$, where ϵ is the least possible amount of energy decrease achievable in one step [20].

Note that the bound of Theorem 2.1 is polynomial in the total weight of the network, and thus networks with small weights always converge in time that is polynomial in the number of nodes. However, if numbers are given in binary representation, then the bound is exponential in the *size* of the representation, and as we shall see in Section 2.2, networks with large weights may indeed require an exponential number of steps to converge. Nevertheless, simulation results show that in practice the number of changing update steps required for convergence is typically at most slightly greater than the number of elements differing between the input and the output vectors [16, 17]. This phenomenon has also been explored analytically by Komlós and Paturi [56], who showed that if a network has been constructed by storing at most $m = n/4\ln n$ random vectors using the tensor product storage rule, then with high probability each of the stored vectors will be surrounded by an attraction domain of radius ρn where $\rho > 0.024$, such that any vector within this domain will converge to the stable vector in $O(\log\log n)$ changing update steps.

Corresponding results can be obtained also for fully parallel operation.

Theorem 2.2 *Let e_i be 1 if $\sum_j w_{ij} - h_i$ is even, and 0 otherwise. Any fully parallel computation of a Hopfield network with integer weights reaches either a stable vector or a cycle of length two in time at most $\frac{1}{2}\Big(\sum_{i,j}|w_{ij}| + 3\sum_i|h_i - e_i| - n\Big)$.*

NOTE: Contrary to sequential operation, the results for fully parallel operation do not require the network to be semisimple.

PROOF: Using the energy function $E(x(t), x(t-1)) = -x(t)^{\mathrm{T}} W x(t-1) + (h-e)^{\mathrm{T}}(x(t) + x(t-1))$, we get the following possibilities: either $x(t+1) = x(t-1)$ (i.e., we have reached a stable vector or a cycle of length two), or $x(t+1) \neq x(t-1)$, which means that the energy difference $E(x(t+1), x(t)) - E(x(t), x(t-1)) \leq -2$.

We see that $1 \leq x_i(t)[\sum_j w_{ij} x_j(t-1) - h_i + e_i] \leq \sum_j |w_{ij}| + |h_i - e_i|$. Summing over i we get $n \leq \sum_{i,j} w_{ij} x_i(t) x_j(t-1) - \sum_i (h_i - e_i) x_i(t) \leq \sum_{i,j} |w_{ij}| + \sum_i |h_i - e_i|$, thus giving $-\sum_{i,j} |w_{ij}| - 2\sum_i |h_i - e_i| \leq E(x(t), x(t-1)) \leq -n + \sum_i |h_i - e_i|$. From this we get $\sum_{i,j} |w_{ij}| + 3\sum_i |h_i - e_i| - n$ as an upper bound on the possible energy difference. The bound follows. $\square$

Already Poljak and Sůra showed that only cycles of length two may appear in fully parallel updates [72].

The result in Theorem 2.2 was obtained by Goles [27] when restricted to $h = \overline{0}$. He also gave networks with binary connection matrices W (and $h = \overline{0}$) that achieve this bound on the convergence time. If the connection matrix is positive definite on the set $\{-1, 0, 1\}^n$, the network always converges to a stable vector [27, 28]. For real-valued weights and thresholds, the bound is $(\frac{1}{2}\sum_{i,j} |w_{ij}| + \sum_i |h_i|)/\epsilon$, where ϵ is the minimal energy decrease in one step [28]. Generally, the bound in Theorem 2.2 is again a worst-case upper bound, and simulation results show very fast convergence [16, 17].

A somewhat weaker version of Theorem 2.2 can also be derived as a corollary to Theorem 2.1 by a most useful trick of "doubling" the network [10, 11, 82]. Assume that a network of n nodes with weights w_{ij} and thresholds h_i is given as in Theorem 2.2. Construct a new, bipartite symmetric network of $2n$ nodes with weights w'_{ij}, where $w'_{i,n+j} = w'_{n+i,j} = w_{ij}$ for $i, j = 1, \ldots, n$, and $w'_{ij} = 0$ otherwise. The thresholds in the new network are defined as $h'_i = h'_{n+i} = h_i$ for $i = 1, \ldots, n$. The idea of the construction is to have the nodes $i = 1, \ldots, n$ of the new network represent the original network at odd points of time, and the nodes $i = n+1, \ldots, 2n$ at even points of time. Note that the only nonzero edges in the new network go from one side to the other. Fully parallel updates of the original network can now be simulated by updating the nodes in the doubled network sequentially in numerical order, since the updating of the nodes $1, \ldots, n$ does not interfere with their inputs from nodes $n+1, \ldots, 2n$, and vice versa. Each round of

Table 1: The convergence behavior of different types of Hopfield networks in different operation modes.

	sequential operation	fully parallel operation
general Hopfield	cycle is possible	stable vector or cycle of length 2
semisimple Hopfield	stable vector	

sequential updates in the doubled network corresponds to two fully parallel update steps in the original network.

According to Theorem 2.1, the sequential updates of the doubled network converge to a stable state in a number of changing update steps at most

$$\frac{1}{2}\sum_{i=1}^{2n}\sum_{j\neq i}|w'_{ij}| + \sum_{i=1}^{2n}|h'_i - e'_i| \quad = \quad \sum_{i=1}^{n}\sum_{j\neq i}|w_{ij}| + 2\sum_{i=1}^{n}|h_i - e_i|.$$

This corresponds to the original network under fully parallel updates converging, within the same time bound, to either a stable state or to a cycle of length two, depending on whether the two sides of the doubled network in its stable state are equal or not equal.

The convergence behavior of the Hopfield network is summarized in Table 1.

2.2 Lower Bounds on the Convergence Time

The worst-case upper bounds on the convergence time of a Hopfield network presented in Section 2.1 are quite pessimistic, and as practical experience and the probabilistic results of Komlós and Paturi [56] show, in a network used as a lightly loaded associative memory, the convergence is typically much faster. In the worst case, however, the upper bounds can be reached; specifically, for networks with large weights and thresholds, convergence may require time that is exponential in the number of nodes in the network. This fact follows in a general form from the results of Schäfer and Yannakakis [76], who showed that the problem of finding a stable vector of a Hopfield network is $\mathcal{PLS}$*-complete*, and hence the

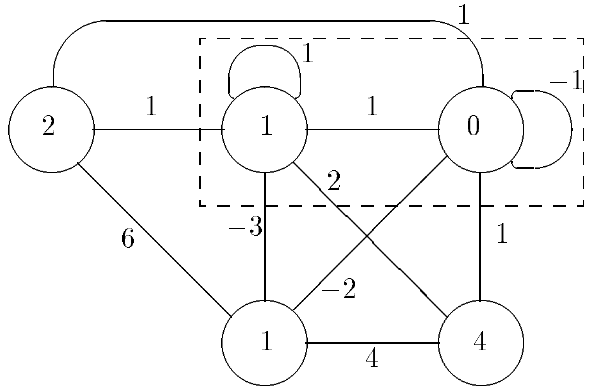

Figure 3: A three-bit fully parallel binary counter network.

standard local improvement algorithm requires exponential time in the worst case. (For more details on this approach, see Section 5.2.) Here we present two explicit constructions: first a network due to Goles and Martínez [29, 30] that requires an exponential number of fully parallel steps to converge, and then a more involved construction by Haken [32] of a network whose convergence time is exponential also in sequential operation[3].

Let us first consider fully parallel updates [29]. For a given n, we construct a network of size linear in n that functions as an n-bit binary counter, and thus takes time at least 2^n to converge. The construction is most easily presented in terms of binary (0/1-valued) nodes, but the networks can of course be translated to the bipolar ($\pm$1-valued) model in the standard manner. Figure 3 presents the first two stages of the construction. The n nodes in the upper row count from all 0's to all 1's (in the figure, the least significant bit is to the right). For each "counter" node added to the upper row, after the two initial ones, two "control" nodes are added to the lower row. The purpose of the latter is to first turn off all the "old" nodes, when the new counter node is activated, and from then on

[3]The history of these lower bounds is quite convoluted: exponential convergence times for fully parallel operation were apparently first shown by Goles and Olivos in [31], although in that case for networks with negative diagonal weights. Haken and Luby [33] seem to have been the first to explicitly observe exponential convergence times for simple networks under a particular sequential update order, although that result in fact already follows from the result of Goles and Olivos by network doubling. With the constructions presented here, and the $\mathcal{PLS}$-completeness result of [76], the issue now seems to be quite conclusively settled.

balance the input coming to the old nodes from the new nodes, so that the old nodes may resume counting from zero[4]. More precisely, the counter node for bit k ($k = 0, 1, 2, \ldots$) is connected to all lower-order counter nodes by interconnections of weight 1, and has threshold k. The first control node of order k is connected to all the lower-order nodes by negative interconnections that have absolute values sufficiently large to overwhelm the positive support these nodes, together with the new counter node, lend to each other. The connection weights from the second control node of order k, on the other hand, are positive, and equal to the absolute values of the weights from the first control node minus one (to compensate for the weight from the counter node of order k). The weights from the counter node of order k to the first associated control node, and from the first control node to the second one must be sufficiently large to drive the updating. (This in particular implies that the weight from the counter node to the first control node must be larger than the sum of all the lower-order weights in the network, and hence exponentially large in k.) It can be verified that an n-bit counter can thus be constructed out of $3n - 4$ nodes, and requires $2^n + 2^{n-1} - 3$ update steps to converge, when started in the all 0's state. For the exact details, see [29] or [30, pp. 88–95].

Let us then turn to the more complicated case of sequential updates. The basic idea of the construction we present, due to Haken [32], is again to implement a binary counter, but the control of the network is now complicated by the necessity of preparing for all possible update orders. This time we work directly in the bipolar model.

The basic unit of the design is the XOR subnetwork presented in Figure 4. It can be seen that the output node of this device eventually obtains the value 1 if the input nodes x and y are clamped in opposite states, and value -1 otherwise.

The counter is constructed by connecting such XOR units in a sequence as illustrated in Figure 5. Each bit b_i, $i = 0, \ldots, n$, of the counter is represented by a pair of nodes x_i, y_i, with the interpretation that b_i has value 1 if $x_i = y_i$. The network counts from an initial counter state of all 1's via the all 0's state back to the all 1's state before converging. The

[4] Following the construction in [29], we have made use of one negative self-loop in the counter network. If desired, this can be removed by making two copies of the least significant node, 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 node. All the other weights and thresholds in the network must then be doubled.

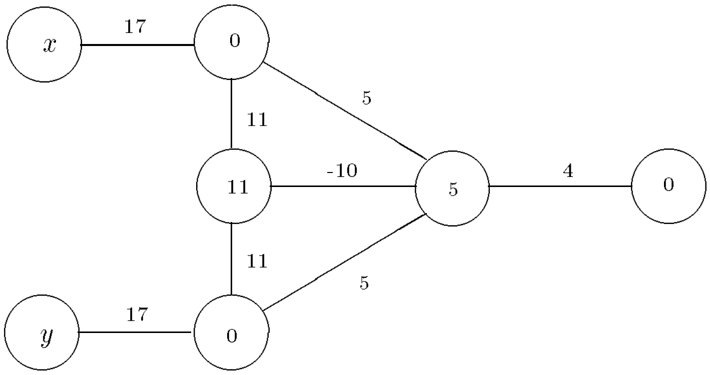

Figure 4: A network gadget for computing the XOR of nodes x and y.

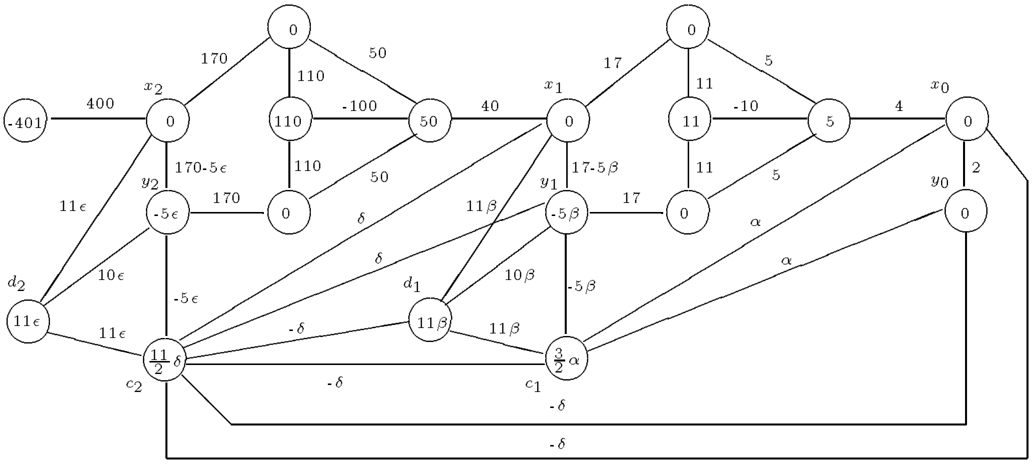

Figure 5: A three-bit sequential counter network.

initial state is represented by $x_i = y_i = -1$ for all i, and the final state by $x_n = y_n = 1$, and $x_i = y_i = -1$ for all $i < n$. In the figure, the high-order nodes are to the left.

The counting proceeds in the following manner (cf. with the illustration): first the network is primed with a signal that starts at the leftmost node in the upper row and moves to the right, changing the state of each x_i node from -1 to 1. This corresponds to resetting each bit b_i from 1 to 0. At the end of the line, also node y_0 gets set to 1, representing bit b_0 being set back to 1.

Subsequent to this initialization, the behavior of the network obeys the following rules: the state of each node y_i gets incremented from -1 to 1 (i.e. bit b_i gets value 1), when $x_{i-1} = y_{i-1} = 1$ and $x_j = y_j = -1$ for all $j < i - 1$. This change at y_i causes, via the XOR line, x_{i-1} being reset back to -1, and each x_j, $j < i - 1$, being set to 1 (i.e. all the lower order bits b_j, $j \leq i - 1$, are reset from 1 to 0). Eventually, again, also y_0 gets value 1, and the counting resumes at the low order bits. Then when counting reaches back to bit b_{i-1}, this time with $x_{i-1} = -1$, $y_{i-1} = 1$, and $x_j = y_j = -1$ for each $j < i - 1$, node y_{i-1} gets decremented from 1 back to -1 (thus setting b_{i-1} to 1), and again each of the lower order x_j's gets set to 1 (corresponding to each b_i getting reset to 0).

To achieve this relatively complicated sequencing, each pair of counter nodes x_i, y_i, $i \geq 1$, has two associated control nodes c_i, d_i, all initially -1. The purpose of node c_i is to test for the first condition above, for turning node y_i from -1 to 1, and the purpose of node d_i is to maintain y_i at 1 until the second condition, for turning it back to -1, is achieved. The functioning of this control structure is perhaps most easily understood by simply simulating the behavior of the sample network in Figure 5. The weights $\alpha \gg \beta \gg \delta \gg \epsilon$ need to be chosen so that control information can only flow from right to left, i.e., that no combination of states to the left of some control node can have an effect on the updates at that node. Denoting $\alpha = \alpha_1$, $\delta = \alpha_2$, $\beta = \beta_1$, $\epsilon = \beta_2$ etc., Haken [32] suggests the values $\alpha_i = 1/(40n)^{(i-1)}$, $\beta_i = n/(40n)^i$ for an n-bit counter. The number of nodes in an n-bit counter is $8n - 5$.

3 Computational Power

In addition to the specific uses of Hopfield networks as associative memories or as devices for combinatorial optimization, they can also be considered as a general model of computation. The central problem then is to characterize the classes of functions computed by different types of networks. Different results are obtained depending on whether a *single* network processes inputs of all sizes, or whether the networks are permitted to *grow* with increasing input sizes. These *uniform* and *nonuniform* network models are discussed subsequently in Sections 3.1 and 3.2. We consider both asymmetric and symmetric networks in this chapter, but restrict ourselves to networks operating in the fully parallel mode.

3.1 Uniform Networks

As is well known, Kleene [51] characterized the computational power of the McCulloch–Pitts [61] model of neural networks – asymmetric finite networks of binary threshold logic units – as equivalent to finite state machines (see also [63]). A significant factor in this result is that only uniform networks were considered. As pointed out by Hartley and Szu in [36], if in addition the networks are restricted to being symmetric, then they do not have even the power of finite automata. (This follows from the strong convergence properties of symmetric networks, as contrasted with finite automata.)

Recently, Siegelmann and Sontag [79] have continued this line of work to networks of nodes with *real-valued* states, where each node computes the saturated-linear function

$$\sigma(x) = \begin{cases} 0 & \text{if } x < 0, \\ x & \text{if } 0 \le x \le 1, \\ 1 & \text{if } x > 1 \end{cases}$$

of its weighted sum of inputs. They show that an asymmetric network of 1058 such nodes is capable of simulating a universal Turing machine. The construction makes use of the well known technique (see, e.g., [40, p. 171]) of representing the tape of a Turing machine by two opposing stacks. Each of the stacks is then represented by the state of one real-valued node; the remaining nodes are used to implement the finite-state control of the universal machine, together with the "push" and "pop" operations of the stacks.

(A similar construction for Turing machine simulation was suggested by Hartley and Szu in [36], but in that case without any of the implementation details.)

A clever idea assisting in the implementation of the stack operations in [79] is to represent binary numbers in a base-4 notation, so that a binary sequence $b_1 \ldots b_n \in \{0,1\}^n$ is represented as $a_1 \ldots a_n \in \{1,3\}^n$, $a_i = 1 + 2b_i$. The sequence is then further coded as the rational number

$$q = \sum_{i=1}^{n} \frac{a_i}{4^i}.$$

If q represents the contents of a stack (the empty stack is represented by 0), then pushing a value $b \in \{0,1\}$ as the first element of the stack can be implemented simply by changing q to $\frac{1}{4}q + \frac{b}{2} + \frac{1}{4}$. Conversely, removing a value b from the top of the stack is achieved by changing q to $4q - 2b - 1$. The top element of a nonempty stack can be computed as $b = \sigma(4q - 2)$. All of these operations can be implemented easily in the network model of real-valued nodes and saturated-linear transfer functions.

Simulations of arbitrary Turing machines by networks with a potentially infinite number of nodes have been suggested by Hartley and Szu [36], and by Franklin and Garzon [21].

3.2 Nonuniform Networks

We now proceed to consider computation by nonuniform sequences of networks. In analogy with standard Boolean circuit complexity theory [70, 86], we assume that some set of nodes in a network are designated as input nodes, on which the input is initially loaded; then the states of the nodes in the network are updated repeatedly, in fully parallel mode, until the network (possibly) converges to some stable global state, at which point the output is read from a set of designated output nodes.

For simplicity, we shall consider here only networks with a single output node; the extensions to networks with multiple outputs are straightforward. The *language recognized* by a single-output network N, with n input nodes, is defined as

$$L(N) = \{x \in \{0,1\}^n \mid \text{ on input } x,\ N \text{ eventually converges with output } 1\}.$$

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

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

Let $\langle x, y \rangle$ be some standard pairing function mapping pairs of binary strings to binary strings (see, e.g. [8, p. 7]). A language $A \subseteq \{0,1\}^*$ belongs to the nonuniform complexity class $\mathcal{P}$/poly, if there are a polynomial time bounded Turing machine M, and an "advice" function $f : N \to \{0,1\}^*$, where for some polynomial q and all $n \in N$, $|f(n)| \leq q(n)$, and for all $x \in \{0,1\}^*$,

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

It is known that the class $\mathcal{P}$/poly equals the class of functions computed by polynomial size bounded sequences of Boolean circuits ([8, p. 100], [49]). The class $\mathcal{PSPACE}$/poly is defined analogously, using polynomial space bounded instead of time bounded Turing machines. For circuit complexity characterizations of this class, see [7].

3.3 Computational Power of Asymmetric Nonuniform Networks

We begin by considering general asymmetric networks.

Since a cyclic network of n nodes converging in time t may be "unwound" into an acyclic network of size $n \cdot t$, the class of Boolean functions computed by polynomial size, polynomial time asymmetric networks coincides with the circuit complexity class $\mathcal{P}$/poly. What is more interesting is that when the computation times of the networks are not bounded, then polynomial size asymmetric networks compute exactly the functions in the space-bounded class $\mathcal{PSPACE}$/poly. The result is not too hard to prove and may be folklore; one of the first printed mentions of it seems to be in an unpublished technical report by Lepley and Miller [59].

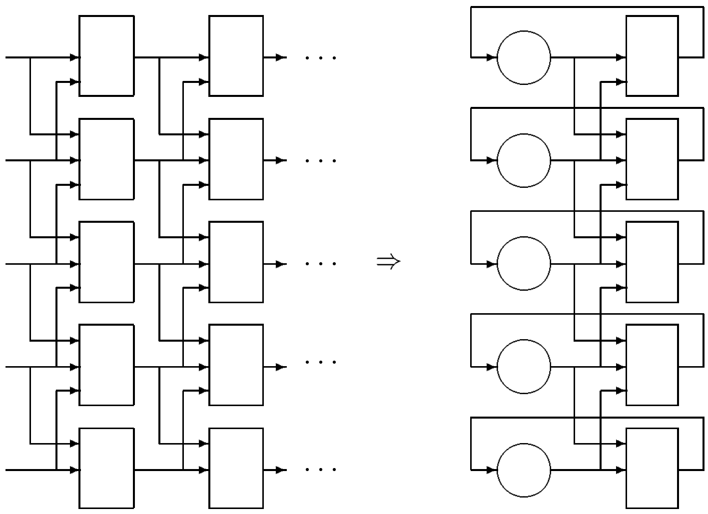

Figure 6: Simulation of a space bounded Turing machine by an asymmetric cyclic network.

Theorem 3.1 *$\mathcal{PNETS} = \mathcal{PSPACE}$/poly.*

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

To prove the converse inclusion, let $A \in \mathcal{PSPACE}$/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. [8]) 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 [8, pp. 106–112], it is straightforward to construct for each n an *acyclic* network that simulates the behavior of M on inputs of length n. (More precisely, the network simulates computations $M(\langle x, f(n)\rangle)$, where $|x| = n$.) This network 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 6, 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 network 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 acyclic circuit by a single transformation layer that feeds back on itself (Figure 6, right). The size of the cyclic network thus obtained is then only $O(q(n))$. When initialized with input x loaded onto the appropriate input nodes, and advice string $f(n)$ mapped to the appropriate initially active nodes, the network will converge in $O(c^{q(n)})$ update steps, at which point the output can be read off the node corresponding to the first square of the machine tape. $\square$

As a corollary to the proof construction, we observe the following result.

Corollary 3.2 *The problem of deciding whether a given fully parallel asymmetric network converges to a stable state from a given initial state is $\mathcal{PSPACE}$-complete.*

NOTE: This problem was considered earlier by Porat [73], who observed that it "does not seem to be in $\mathcal{P}$". She proved that the problem of deciding whether an asymmetric network converges from all initial states is $\mathcal{NP}$-hard.

PROOF: Since a given network can be simulated *in situ*, the problem is clearly in $\mathcal{PSPACE}$. To prove completeness, let A be an arbitrary language in $\mathcal{PSPACE}$, and let M be a polynomial space bounded Turing machine recognizing $\bar{A}$, the *complement* of A. Without loss of generality, we may assume that M halts on all inputs, and prints a 1 on the first square of its tape if and only if it accepts its input. Now given a string x, construct as above an asymmetric network simulating the behavior of M on input x, with the extra property that the network enters a cycle if ever a 1 appears on the wire corresponding to the first square of the machine tape. (This can be achieved by a simple subnetwork connected to that wire.) Clearly this transformation can be computed in polynomial time, and $x \in \bar{A}$ if and only if the constructed network does not converge, i.e., $x \in A$ if and only if the network converges. Hence the transformation is a polynomial time reduction from the problem of recognizing A to the convergence problem of asymmetric networks. $\square$

3.4 Computational Power of Symmetric Nonuniform Networks

The straightforward approach to extending the previous simulation to work on symmetric networks would be to simulate each of the asymmetric edges in the constructed network by a subnetwork of symmetric ones. This is not possible in general, as is shown by the different convergence behaviors of asymmetric and symmetric networks. However, in the special case of *convergent* computations of asymmetric networks the simulation can be achieved [67].

Theorem 3.3 *$\mathcal{PNETS}$(symm) = $\mathcal{PSPACE}$/poly.*

PROOF: Because $\mathcal{PNETS}(\text{symm}) \subseteq \mathcal{PNETS}$, and by the previous theorem $\mathcal{PNETS} \subseteq \mathcal{PSPACE}/\text{poly}$, it suffices to show the inclusion $\mathcal{PSPACE}/\text{poly} \subseteq \mathcal{PNETS}(\text{symm})$.

Given any $A \in \mathcal{PSPACE}/\text{poly}$, 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 nodes and e edges of nonzero weight on a symmetric network of $O(n+e)$ nodes and $O(n^2)$ nonzero edges.

The construction combines two network "gadgets": a simplified version of a mechanism due to Hartley and Szu [36] for simulating an asymmetric edge by a sequence of symmetric edges and their interconnecting nodes, whose behavior is coordinated by a system clock (Figures 7, 8); and the parallel binary counter network of Goles and Martínez [29] discussed in Section 2.1. An important observation here is that any convergent computation of an asymmetric network of n nodes has to terminate in 2^n synchronous update steps, because otherwise the network repeats a configuration and goes into a loop; hence, the exponential time counter network can be used to provide a sufficient number of clock pulses for the

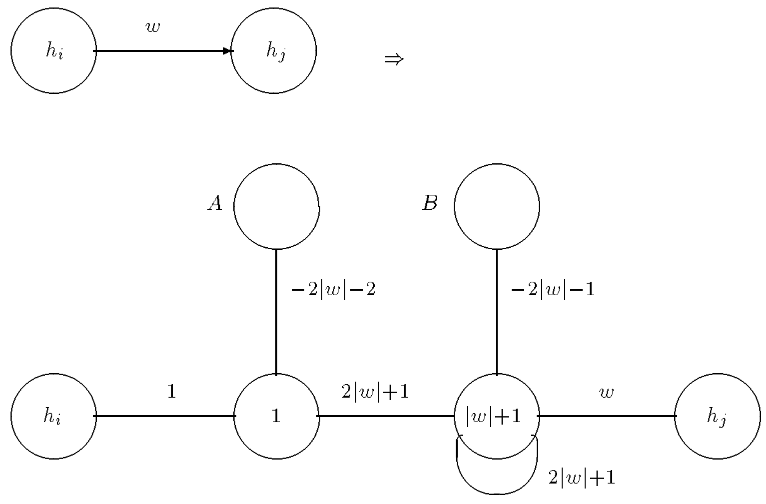

Figure 7: A sequence of symmetric edges simulating an asymmetric edge of weight w.

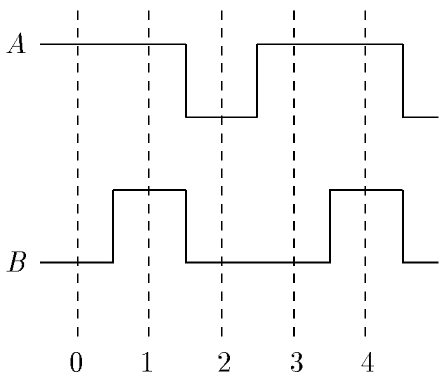

Figure 8: The clock pulse sequence used in the edge simulation of Figure 7.

simulation to be performed.

Let us first consider the gadget for a symmetric simulation of an asymmetric edge of weight w from a node i to a node j (Figure 7) [36]. Here the idea is that the two nodes inserted between the nodes i and j in the synchronous 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 nodes labeled A and B, in cycles of length three as presented in Figure 8.

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

The A and B pulse sequences required for the simulation may be derived from the Goles–Martínez counter network (Figure 3) by means of a delay line network such as the one presented in Figure 9. The oscillator node shown in the upper left corner of the delay line corresponds to the second counter node in the Goles–Martínez network, which is "on" for four update steps at a time. The *second* counter node is chosen because the A and B sequences are of period three, and the oscillator has to be slower than that. (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 node turns on, it "primes" the first node 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 node per update step, and the pulses A and B are derived from the appropriate points in the line.

The value K used in the construction has to be chosen so large that the states of the nodes in the underlying network have no effect back on the delay line. It is sufficient to choose K larger than the total weight of the underlying network. Similarly, the weights

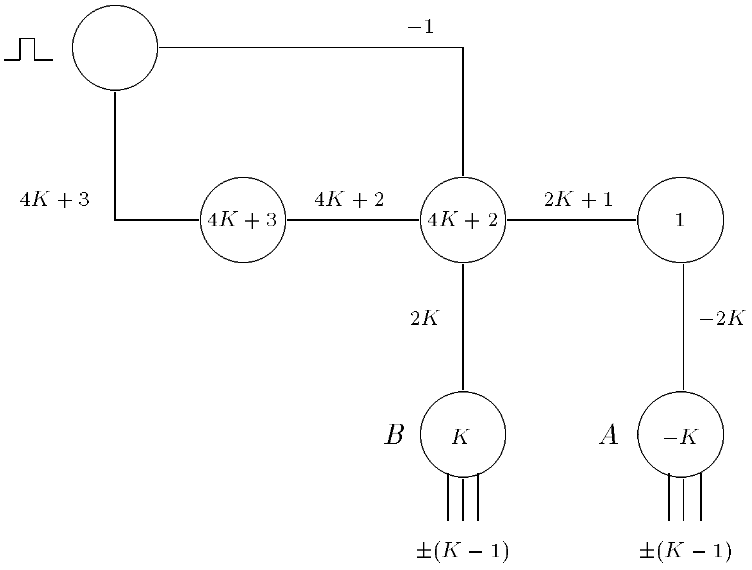

Figure 9: A delay line for generating clock pulses from the binary counter network in Figure 3.

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 $K \geq 3$, it is here sufficient to multiply all the weights and thresholds by $6K$, and then subtract one from each threshold. $\square$

Concerning the edge weights in the above constructions, one can see that in the network implementing the machine simulation (Figures 6, 7), 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 3) 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 3.4 *$\mathcal{PNETS}$(symm, small) = $\mathcal{P}$/poly.*

Consequently, anything that can be computed by asymmetric networks in polynomially bounded time can also be computed by polynomial weight symmetric networks which are guaranteed to have the good convergence properties discussed in Section 2.

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 $\mathcal{P}/\text{poly} \neq \mathcal{PSPACE}/\text{poly}$. (The condition $\mathcal{P}/\text{poly} = \mathcal{PSPACE}/\text{poly}$ is known to have the unlikely consequence of collapsing the "polynomial time hierarchy" to its second level [49].) In asymmetric networks large weights are not needed, in the sense that any node with large incoming weights can be replaced by a polynomial size subnetwork of depth 2 [25].

4 Synthesis of Nets for Associative Memory

The most extensively studied application of Hopfield nets is their use as associative memories. Many different storage rules have been proposed and analyzed in the literature. We shall touch on this fascinating area only very briefly here; for much more information, see e.g. the section on associative memories by Paturi in the current volume, or the book by Kamp and Hasler [46].

A set $u^1, \ldots, u^m$ of bipolar vectors is *stored* in a Hopfield net with weight matrix W and threshold vector h if the vectors correspond to stable states of the global update rule of the network, i.e., if $\mathrm{sgn}(Wu^p - h) = u^p$ for all $p = 1, \ldots, m$. The set of all vectors that are guaranteed to eventually converge to a given stable vector u forms the *attraction domain* of u[5]. The *attraction radius* of a stable vector u is the largest Hamming distance from within which all vectors are guaranteed eventually to converge to u. The *k-step* attraction radius is the largest distance from within which all vectors converge to u in at most k update steps.

A *storage rule* is an algorithm for constructing, from a given set of vectors u^i, a Hopfield net that stores those vectors (or at least a net that with high probability stores most of the u^i approximately correctly). It is naturally also desirable that the stored u^i have nontrivial attraction radii.

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 n, the number of nodes 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.

[5] Note that while the set of stable vectors of a network does not depend on the update rule, a given unstable vector can in sequential operation converge to different stable vectors, depending on the update order. For simplicity, we shall only consider fully parallel updates here, but in the definitions for sequential updates convergence should be required *irrespective* of the update order chosen. Also note that while in sequential operation the attraction domains are contiguous, in fully parallel operation they can consist of disjoint regions far from each other.

4.1 Synthesis Methods

The most widely studied storage rule (but probably not the best one to use) is the *tensor product*, or *outer product* rule, whereby the weight matrix W and threshold vector h are chosen as $W = \sum_{i=1}^{m}(u^i(u^i)^{\mathrm{T}} - I)$ and $h = \overline{0}$. The rule expresses a Hebbian [37] way of thinking: If two elements of a stored vector are identical, there should be a positive connection between the corresponding neurons, and if the elements are opposite, the connection should be negative. The diagonal of W contains only zeros, i.e., the network is simple. The tensor product rule was studied in the 1970's by Anderson [4], Kohonen [52], Nakano [66], and others, and then popularized by Hopfield in his influential 1982 paper [41].

In relation to the tensor product rule, Baum et. al. [9] and Amari [2] have studied the advantages of *sparse encoding*, which means that only a small fraction of the bits are $+1$ (or -1). The vectors can be preprocessed so that they become almost orthogonal or are made longer and sparse — but this preprocessing naturally costs space and time.

Alternatives to the tensor product rule are the so called *spectral* or *pseudoinverse algorithms* discussed, for instance, in [15, 71, 84]. The basic idea, which can again be traced back to the work of Kohonen et al. in the 1970's [54], is to construct the weight matrix W so that the stored vectors, if they are linearly independent, become eigenvectors of W. This can be achieved by choosing, e.g., $W = U(U^{\mathrm{T}}U)^{-1}U^{\mathrm{T}}$, where U is a column matrix of the vectors $u^1, u^2, \ldots, u^m$. The method obviously guarantees perfectly noiseless retrieval of any set of linearly independent vectors.

In general, the problem of deciding whether a given set of vectors can be stored as stable states of a Hopfield network can be solved in polynomial time by linear programming. Given the vectors $u^1, u^2, \ldots, u^m \in \{0,1\}^n$, set up the following system of linear inequalities in the variables w_{ij}, $i,j = 1,\ldots,n$ and h_i, $i = 1,\ldots,n$:

$$\begin{array}{lll} \text{(i)} & w_{ij} = w_{ji}, & \text{for } i,j = 1,\ldots,n, \quad i \neq j; \\ \text{(ii)} & w_{ii} = 0, & \text{for } i = 1,\ldots,n; \\ \text{(iii)} & \sum_{j=1}^{n} u_j^p w_{ij} - h_i \begin{cases} \geq 0, & \text{if } u_i^p = 1 \\ < 0, & \text{if } u_i^p = -1 \end{cases}, & \text{for } i = 1,\ldots,n, \quad p = 1,\ldots,m. \end{array}$$

This system of $O(mn^2)$ inequalities can then be solved by, e.g., some variant of the ellipsoid method (see [77]). This approach gives no control over the attraction radii of the vectors;

however, Chandru et. al. [13] propose a modification of the technique whereby the *direct* attraction radii of the vectors can be maximized in the solution. (For definitions and discussion of the problem of determining attraction radii in Hopfield nets, see Section 5 below.)

4.2 Capacity Issues

As an upper bound on the capacity of any storage rule one can define the *optimal* capacity of a network to be the maximum number of vectors that can be stored, when the weight matrix and threshold vector can be chosen freely. It is known that if no requirements are set on the attraction radii of the stored vectors, then with high probability up to $2n$ random vectors may be stored in a network with asymmetric connections [14, 22, 83, 85]. However, when the stored vectors are not random, but are chosen in the worst possible manner, the optimal capacity is much lower.

Theorem 4.1 *[12, 64] The worst-case capacity for Hopfield networks with at least one zero element in the diagonal of the connection matrix is 1.*

PROOF: Clearly, at least one vector can be stored. On the other hand, assume that element j in the diagonal of the connection matrix is zero. Take any vector u^1 and as the other vector u^2, which differs only in component j. But then the j^{th} component of $Wu^1 - h$ and $Wu^2 - h$ are the same, and u^1 and u^2 cannot both be stable vectors at the same time. $\square$

For an approach to estimating the probabilistic capacity of the tensor product storage rule, consider the following "signal-to-noise ratio" calculation. If the weight matrix W is formed according to the tensor product rule from the vectors $u^1, \ldots, u^m$, then for every vector u^i, and every index r, $(Wu^i)_r = (n-1)u^i_r + \sum_{j \neq i} \sum_{s \neq r} u^i_s u^j_s u^j_r$. If the components of the vectors u^i are uniform i.i.d. random variables (i.e., different elements are -1 or $+1$ with equal probability independently of each other), then the first "signal" term in the sum equals $(n-1)u^i_r$, while the second "noise" term obeys a binomial distribution with zero mean and variance $(n-1)(m-1)$. This means that as long as m is small, we can expect the stored vectors to be stable [41, 62]. In situations with many long stored vectors, this distribution can be approximated by a Gaussian distribution with the same mean and variance.

For random vectors stored with the tensor product storage rule, the probability that *all* vectors are perfectly recalled goes to 1, if the number of stored vectors is at most $n/(4 \ln n)$; the probability that *almost all* vectors are perfectly recalled goes to 1, if the number of stored vectors is at most $n/(2 \ln n)$ [58, 62, 84]. If the number of stored vectors exceeds $0.138n$, an avalanche phenomenon can occur [3]: The errors increase in each update step, and although the first update step results in a vector not far from the desired one, the final result is far from correct. Thus, we can store at most $0.138n$ vectors if we are willing to accept the errors that occur up to that point.

If the stored vectors are almost orthogonal, then clearly the "noise" term is very near to zero. This type of restriction has been studied, for instance, by Dembo [15]. He proves that if the Hamming distances between each pair of stored vectors are between αn and $(1-\alpha)n$ for fixed α, $0 < \alpha < 1/2$, then the worst-case capacity for Hopfield networks with at least one zero element in the diagonal of the connection matrix is at most $\lceil 1/(1-2\alpha) \rceil$, and the capacity approaches $\lceil 1/(1-2\alpha) \rceil$, as $n \to \infty$.

It has been proved by Komlós [55] that the probability of n random binary vectors being linearly independent approaches 1 as $n \to \infty$. It follows immediately that the probabilistic capacity of the spectral algorithms is n, when only the stability of the stored vectors is considered, with no requirements on the attraction radii.

4.3 Complexity of the Synthesis Problem

Surprisingly, despite all the work on the design and analysis of storage rules for Hopfield nets, the exact computational complexity of the synthesis problem is still unknown, i.e., it is not known whether the following problem is polynomial time solvable:

> HOPFIELD NET SYNTHESIS:
> Given an integer ρ and m vectors $u^1, u^2, \ldots, u^m \in \{-1, 1\}^n$, such that the distance between each vector pair (u^i, u^j) (where $i \neq j$) is greater than 2ρ; is there an integer matrix W and an integer threshold vector h, so that all vectors u^i are stable and have an attraction radius of (at least) ρ?

As pointed out above, for $\rho = 0$ the problem can be solved in polynomial time by linear programming, but even the case $\rho = 1$ is open. For fixed ρ and fully parallel operation the problem is in $\mathcal{NP}$, provided that only networks with small weights are considered. For unbounded ρ, it is not even clear that the problem is in $\mathcal{NP}$. (With the small weights restriction, it can be seen to be in the class Σ_2^p of the polynomial time hierarchy [7].)

5 Computational Problems in the Analysis of Hopfield Nets

Assume that we have stored the vectors $u^1, u^2, \ldots, u^m \in \{-1,1\}^n$ in a Hopfield associative memory. This means that we have determined a connection matrix W and a threshold vector h somehow, so that all the vectors u^1, u^2, $\ldots$, u^m are stable. In this section we examine how difficult it is to assess the quality of the resulting network by the means of a conventional computer (classical complexity classes).

The symmetry condition of the connection matrix is important: Analyzing even the very basic properties of networks with asymmetric connection matrices is hard. For instance, as was pointed out in Corollary 3.2, it is $\mathcal{PSPACE}$-complete to decide whether a given computation by an asymmetric network converges to a stable vector, and Alon and Lipscomb [1, 60] have shown that even the problem of determining whether such a network *has* any stable vectors is $\mathcal{NP}$-complete.

We analyze the complexity of determining the number of stable vectors in a given network, and the complexity of computing the sizes of their attraction domains and radii. Most of the problems we examine are hard even for simple networks, hence also for more general networks. Whenever possible, we try to state the results using simple networks.

We shall need the following complexity-theoretic notions. An integer valued function f belongs to the class $\#\mathcal{P}$ if there is a nondeterministic polynomial time Turing machine M that on each input x has exactly $f(x)$ accepting computation paths. A function f is *$\#\mathcal{P}$-complete*, if it is in $\#\mathcal{P}$, and any other function in $\#\mathcal{P}$ can be computed by some deterministic polynomial time Turing machine that is allowed to access values of f at unit cost. To each $\mathcal{NP}$ decision problem there corresponds in a natural way a $\#\mathcal{P}$ counting problem (the problem of counting accepting computation paths, i.e., alternative solutions to a given problem instance), and vice versa. We shall say, somewhat inaccurately, that an $\mathcal{NP}$ decision problem A is in $\#\mathcal{P}$ (respectively $\#\mathcal{P}$-complete) if the associated counting problem is in $\#\mathcal{P}$ (respectively $\#\mathcal{P}$-complete). For a more extended discussion of these notions, see [23, pp. 168–169].

5.1 Number of Stable Vectors

Given a network with connection matrix W and threshold vector h, it is easy to check that the stored vectors u^i, for $i = 1, 2, \ldots, m$, are all stable: Just compute $\text{sgn}(Wu^i - h)$ for each one. Hence, computing the total number of stable vectors in a network is equivalent to computing the number of *spurious* stable vectors it has.

It is an $\mathcal{NP}$-complete problem to decide whether there are any stable vectors in a Hopfield network with negative diagonal elements allowed [18]. Nevertheless, even in simple networks it is hard to determine the *number* of stable vectors, as the following result shows.

Theorem 5.1 *[60] The problem "Given a simple Hopfield network with small nonpositive connection weights and small threshold values; are there two stable vectors in the network?" is $\mathcal{NP}$-complete.*

PROOF: It is clear that the problem is in $\mathcal{NP}$. To prove completeness, we construct a polynomial time reduction from the $\mathcal{NP}$-complete problem of finding large independent sets in graphs (problem IND SET [23, 48]):"Given an undirected graph G with n nodes and an integer k, where $1 \leq k \leq n$; does G contain an independent set of size k, i.e., a set of k nodes no two of which are connected by an edge?"

We construct in the following a polynomial time reduction from IND SET: Consider an instance $\langle G, k\rangle$ of the problem IND SET. Choose numbers α, β, γ, and δ, so that $0 < \alpha < 2\beta$, $0 < \gamma < 2\delta$, and $2(n-k)\delta \leq \alpha < 2(n-k)\delta + 1$. Possible values are $\alpha = 2(n-k)$, $\beta = n - k + 1$, and $\gamma = \delta = 1$. Construct a Hopfield network H so that (cf. Figure 10):

1. for each node v in G, there is a node in H with threshold $-\alpha + \beta d_G(v) - \delta(n-1)$, where $d_G(v)$ is the number of neighbors (i.e., the degree) of node v in G,

2. for each edge in G, there is a connection with weight $-\beta$ between the corresponding nodes in H,

3. for each pair u, v of nodes in G, there is a new node with threshold $-\gamma$ and connections with weights $-\delta$ between the new node and u and between the new node and v.

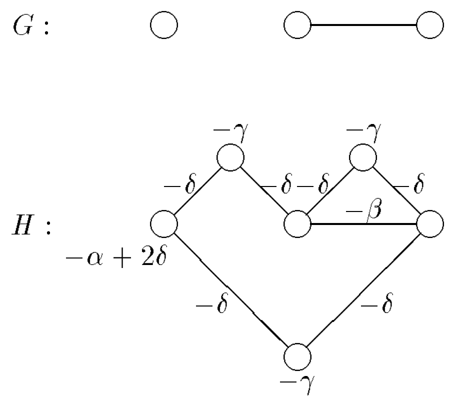

Figure 10: An example of a graph G and the Hopfield network H constructed from it. The two nodes in H with thresholds not written out in the figure have threshold values $-\alpha + \beta + 2\delta$.

The nodes and connections in (1) and (2) are called old nodes and connections; the nodes and connections in (3) are called new nodes and connections. Each old node v has $d_G(v)$ old neighbors and $n-1$ new neighbors. Each new node has two old neighbors and no new neighbors. From the choice of weights and thresholds we get the following properties:

Property 1 For each pair of nodes u, v in G holds that if u, v are 1 and stable in H, then u and v are independent of each other in G.

From this and from the fact that all thresholds are negative it follows that for the old nodes and connections stable vectors in H correspond exactly to maximal independent sets in G.

Property 2 For an old node v it holds that v is 1 and stable if and only if at least $k-1$ of its new neighbors are -1 and each old neighbor is -1.

Property 3 For a new node v holds that v is -1 and not stable if and only if both (old) neighbors are 1.

Additionally, from Properties 2 and 3 we can easily prove the following property:

Property 4 There is a (trivial) stable vector with each old node -1 and each new node 1.

Now we prove that H has two stable vectors if and only of G has an independent set of size k.

Suppose that H has two stable vectors. Then there is a stable vector where some old node is 1 or where some new node is -1 (otherwise only Property 4's stable vector is stable). By Property 3, this means that some old node is 1. Further, by Property 2, at least $k-1$ of the old node's new neighbors are -1, which implies, by Property 3, that at least k old nodes are 1. But Property 1 then says that these nodes are independent; in other words, G has an independent set of size k.

On the other hand, suppose that G has an independent set of size k. Then Property 4's vector is one and the following vector is the other stable vector: Take the independent set of size k and add nodes one at a time so that the result is a maximal independent set. Choose the nodes in this maximal independent set to have state 1. Choose state -1 for the other old nodes. Choose state -1 for the new nodes between a pair of independent old nodes, and choose state 1 for the other new nodes. $\square$

The problem of whether there exist two stable vectors is $\mathcal{P}$-complete, if all connection weights are nonnegative, but $\mathcal{NP}$-complete if all connection weights are nonpositive [60].

The problem with $h = \overline{0}$, but without the restriction to small weights and with three instead of two stable vectors has been proved $\mathcal{NP}$-complete in [17, 18]. This result and Theorem 5.1 can easily be strengthened to $\#\mathcal{P}$-completeness results.

Corollary 5.2 *[17, 18, 60] The problems "Given a simple Hopfield network with threshold vector $h = \overline{0}$; how many stable vectors are there in the network?" and "Given a simple Hopfield network with small nonpositive connection weights and small threshold values; how many stable vectors are there in the network?" are $\#\mathcal{P}$-complete.*

It is known that if in a simple Hopfield network with threshold vector $h = \overline{0}$ the elements of the connection matrix are independent identically distributed zero-mean Gaussian random variables, the asymptotic estimate for the number of stable vectors is about $1.05 \times 2^{0.2874n}$ [62, 81].

5.2 Finding Stable Vectors

We now turn to the problem of finding a stable vector to which a given initial vector may converge under sequential operation.

It is easy to see that a sequential computation by the network itself solves the problem in polynomial time, if the weights are small. Alon [1] has showed that there is an algorithm for finding some stable vector in time $O(n^2)$, if the connection weights are all nonnegative (they need not be small). On the other hand, if all the connection weights are small and nonpositive, it is $\mathcal{P}$-hard to find some stable vector [24].

Local search problems have a neighborhood structure that associates with every solution a set of neighboring solutions. Starting from some initial solution, the algorithm keeps moving to a better neighboring solution, until it arrives at a locally optimal solution, i.e., one that does not have a better neighbor. As already noted in previous sections, the search for a stable vector can be viewed as a search for a local minimum of the energy function $E(x) = -\frac{1}{2}x^{\mathrm{T}}Wx + (h-e)^{\mathrm{T}}x$. In sequential operation, two vectors are neighbors if they differ in exactly one element.

Johnson, Papadimitriou, and Yannakakis introduced in [45] a complexity class called $\mathcal{PLS}$ (for Polynomial-time Local Search). For problems in $\mathcal{PLS}$, we can determine in polynomial time whether a solution is locally optimal and find a better neighbor if it is not.

We now give the formal definitions. Each instance x of a local search problem P has a set of feasible solutions $F(x)$. Each feasible solution $s \in F(x)$ has an integer cost $\mu(s,x)$ that is to be optimized (minimized or maximized) and a set of neighboring solutions $N(s,x)$. A solution s is locally optimal if it does not have a strictly better neighbor.

The local search problem is: "Given input instance x, find a locally optimal solution for x." Now $P \in \mathcal{PLS}$ if there exists three polynomial-time algorithms A, B, and C, defined as follows:

1. Algorithm A on input x produces an initial solution from $F(x)$.

2. Algorithm B on inputs x and $s \in F(x)$ computes $\mu(s,x)$.

3. Algorithm C on inputs x and $s \in F(x)$ either determines that s is locally optimal

or finds a better solution in $N(s, x)$.

A problem $P \in \mathcal{PLS}$ is $\mathcal{PLS}$-reducible to another problem $Q \in \mathcal{PLS}$ if there are polynomial-time computable functions Φ and Ψ such that

1. Φ maps instances x of P to instances $\Phi(x)$ of Q,

2. Ψ maps pairs of the form (solution of $\Phi(x)$, x) to solutions of x, and

3. for every instance x of P, if s is a locally optimal solution for the instance $\Phi(x)$, then $\Psi(s, x)$ is a locally optimal solution for the instance x of P.

$\mathcal{PLS}$ reductions compose, and if locally optimal solutions to Q can be found in polynomial time, then locally optimal solutions to P can be found in polynomial time. The class of $\mathcal{PLS}$-complete problems is defined in the obvious way. A $\mathcal{PLS}$-complete problem can in principle simulate any local search problem of $\mathcal{PLS}$.

The following theorems gather together some basic facts about $\mathcal{PLS}$ [45]. Here $\mathcal{P}_S$ and $\mathcal{NP}_S$ are classes of search problems analogous to the decision classes $\mathcal{P}$ and $\mathcal{NP}$. It is not difficult to show that $\mathcal{P}_S = \mathcal{NP}_S$ if and only if $\mathcal{P} = \mathcal{NP}$.

Theorem 5.3 $\mathcal{P}_S \subseteq \mathcal{PLS} \subseteq \mathcal{NP}_S$.

Theorem 5.4 *If a $\mathcal{PLS}$ problem is $\mathcal{NP}$-hard, then $\mathcal{NP}$ = co-$\mathcal{NP}$.*

Theorem 5.4 implies that it is unlikely that $\mathcal{PLS} = \mathcal{NP}_S$.

The *standard algorithm* finds a locally optimal solution by repeatedly applying algorithm C. The *standard algorithm problem* is the search problem "Given an instance and an initial solution s; find the local optimum that would be produced by the standard algorithm starting from s." Note that the answers to the problem depend in general on the rule for choosing the next solution from the neighbors.

The problem of finding a stable vector for a network with only nonnegative connection weights is solvable in polynomial time. In this case, even a global optimum can be constructed in polynomial time. Furthermore, for small weights, the problem can be solved in Random $\mathcal{NC}$ [50].

The results of Schäffer and Yannakakis [76] can now be summarized in the following theorem that we state without proof. The proofs are quite involved.

Theorem 5.5

1. *To find a stable vector in a network with all connection weights* 0 *or* -1 *is* $\mathcal{P}$*-complete.*

2. *The problem of finding a stable vector is* $\mathcal{PLS}$*-complete. The reductions used (i.e., the functions* Φ *and* Ψ*) are in* $\mathcal{LOGSPACE}$*.*

3. *The standard algorithm takes exponential time in the worst case for any rule for choosing the next vector from the neighbors.*

4. *The standard algorithm problem is* $\mathcal{NP}$*-hard.*

5.3 Attraction Domains of Stable Vectors

In this section we consider the problem of computing the size of the attraction domain of a given vector, i.e., the number of vectors that converge to the given vector. We restrict our study to fully parallel operation.

We show that asking for another vector in the attraction domain of a stable vector is $\mathcal{NP}$-complete. We first state a lemma claiming that a certain very restricted form of 0–1 integer programming is $\mathcal{NP}$-complete.

Lemma 5.6 *[17, 18] The problem "Given a symmetric zero diagonal* $n \times n$*-matrix* A *with entries from* $\{-1, 0, 1\}$*; does there exist a vector* $x \in \{0,1\}^n$*,* $x \neq \overline{0}$*, such that* $Ax \geq \overline{0}$*?" is* $\mathcal{NP}$*-complete.*

The next theorem shows that it is difficult to decide, when using fully parallel operation, whether the direct attraction domain, and thereby the general attraction domain, of $\overline{1}$ is bigger than $\{\overline{1}\}$.

Theorem 5.7 *The problem "Given a simple Hopfield network with stable vector* $\overline{1}$*, with threshold vector* $h = \overline{0}$*, and with* $w_{ij} \in \{-1, 0, 1\}$*; is there any other vector going to* $\overline{1}$ *in one fully parallel step?" is* $\mathcal{NP}$*-complete.*

Proof: The problem can be rephrased as "Given a symmetric zero diagonal matrix W with entries from $\{-1, 0, 1\}$ and such that $\mathrm{sgn}(W\overline{1}) = \overline{1}$; is there any bipolar vector $x \neq \overline{1}$, such that $\mathrm{sgn}(Wx) = \overline{1}$?" The problem is clearly in $\mathcal{NP}$. We prove that it is $\mathcal{NP}$-hard by a reduction from the problem in Lemma 5.6. Consider an instance of that problem: "Given a symmetric zero diagonal $k \times k$-matrix A with entries from $\{-1, 0, 1\}$; does there exist a vector $y \in \{0, 1\}^k$, $y \neq \overline{0}$ such that $Ay \geq \overline{0}$?"

Let K_1 be the symmetric zero diagonal $k \times k$-matrix whose entry k_{ij} is 1 if $|i - j| = 1$, and 0 otherwise. Here the subtractions are taken modulo k. The main property of K_1 is that it has exactly 2 elements with value 1 on every row and every column. Let 1 denote the matrix whose all entries are 1. Choose

$$W = \begin{pmatrix} -A & 0 & 0 & A \\ 0 & -K_1 & I & I \\ 0 & I & 0 & 0 \\ A & I & 0 & 1 - I \end{pmatrix}.$$

It is easy to verify that $\mathrm{sgn}(W\overline{1}) = \overline{1}$. We now show that there exists a vector $y \in \{0, 1\}^k$, $y \neq \overline{0}$, such that $Ay \geq \overline{0}$ in the problem in the lemma if and only if there exists an $x \in \{-1, 1\}^{4k}$, $x \neq \overline{1}$, such that $Wx \geq \overline{0}$ in the present problem. Let us denote $x = (x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)})$, where $x^{(i)} \in \{-1, 1\}^k$, for $i = 1, 2, 3, 4$.

(if) If $Wx \geq \overline{0}$ for some $x \neq \overline{1}$, then $x^{(2)} = \overline{1}$ by the third part of matrix W (i.e., rows $2k + 1, 2k + 2, \ldots, 3k$). Thus $x^{(3)} = x^{(4)} = \overline{1}$ by the second part of W. Hence $A(\overline{1} - x^{(1)}) \geq \overline{0}$ by the first part of W, and $x \neq \overline{1}$. In other words, $Ay \geq \overline{0}$, and $y \neq \overline{0}$, where $y = (\overline{1} - x^{(1)})/2$.

(only if) If $Ay \geq \overline{0}$ and $y \neq \overline{0}$, then take $x = (x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}) = (\overline{1} - 2y, \overline{1}, \overline{1}, \overline{1})$. Now $Wx \geq \overline{0}$, and $x \neq \overline{1}$. This concludes the proof. □

Also, counting the number of vectors going to the stable vector $\overline{1}$ in a single fully parallel step is hard.

Corollary 5.8 *[17, 18] The problem "Given a simple Hopfield network with stable vector $\overline{1}$, with threshold vector $h = \overline{0}$, and $w_{ij} \in \{-1, 0, 1\}$; how many other vectors go to $\overline{1}$ in one fully parallel step?" is $\#\mathcal{P}$-complete.*

5.4 Attraction Radii of Stable Vectors

Finally we confront the attraction radius. The presentation follows that of [19].

We start by examining the easier case: the fully parallel network. As will be seen, the boundary between tractability and intractability is here located between computing the direct (one-step) and the two-step attraction radii. We first observe that the former can be computed in polynomial time.

Theorem 5.9 *The problem "Given a fully parallel Hopfield network, a stable vector u, and a distance k; is the direct attraction radius of u equal to k?" is polynomially solvable.*

PROOF: The following polynomial time procedure determines the direct attraction radius of u:

1. radius := n;
2. for each node i do:
   (a) compute the values $w_{ij}u_j$ for $j = 1, \ldots, n$ and order them as $\alpha_1 \geq \cdots \geq \alpha_n$;
   (b) sum := $\sum_j \alpha_j - t_i$ % this is the total input to node i;
   (c) if $u_i = 1$ then do:
      i. $k := 1$;
      ii. repeat sum := sum$-2\alpha_k$; $k := k + 1$ until sum < 0 or $\alpha_k \leq 0$ or $k = n + 1$;
      iii. if sum < 0 then radius := min{radius, $k - 2$}
   (d) if $u_i = -1$ then do:
      i. $k := n$;
      ii. repeat sum := sum$-2\alpha_k$; $k := k - 1$ until sum ≥ 0 or $\alpha_k \geq 0$ or $k = 0$;
      iii. if sum ≥ 0 then radius := min{radius, $n - k - 1$}
3. return(radius).

Intuitively, we check for each node how many of its inputs must be altered to change its state in the update. The minimum of these numbers is the distance to the nearest vector that results in something else than u. If this distance is k, the radius of direct attraction is $k - 1$. □

Next we consider the problem of computing the general attraction radius. Note that this problem is in $\mathcal{NP}$ if the weights in the network are polynomially bounded in n. A nondeterministic algorithm for the problem works as follows: Given a vector u and a distance k, guess a vector that is within distance k from u and does not converge to u, witnessing that the attraction radius of u is less than k. When the weights are polynomially bounded, any vector converges to either a stable vector or a cycle of length two in polynomial time (Theorem 2.2).

Theorem 5.10 *The problem "Given a fully parallel Hopfield network, a stable vector u, and a distance k; is the attraction radius of u less than k?" is $\mathcal{NP}$-hard.*

PROOF: We prove that the problem is $\mathcal{NP}$-hard by a reduction from the $\mathcal{NP}$-complete satisfiability problem SAT: "Given a conjunctive normal form (CNF) formula F with k variables and m clauses, is there a satisfying truth assignment for F?"[6] In fact, we need a special version of SAT: we require that no clause contains both a variable and its negation, and we require that the number of clauses is greater than the number of variables. It is easy to see that these requirements do not affect the $\mathcal{NP}$-completeness, since clauses with both a variable and its negation can be excluded, and if the number of clauses is too small, we can simply repeat one of the clauses the required number of times.

Let $\tilde{x} = (\tilde{x}_1, \ldots, \tilde{x}_k)$ be some truth assignment not satisfying F. Such an assignment is easy to find: Take, for instance, the first clause and choose values for the variables against their appearance in the clause; i.e., if variable x_i is in the clause, choose `false` for it, if x_i appears negated in the clause, choose `true` for it; otherwise the value of the variable does not affect the value of the clause, so choose for instance value `false` for it.

Transform formula F to an equivalent formula F' by adding k times one of the `false` clauses. This ensures that at least $k+1$ of the clauses evaluate to `false` under $\tilde{x}$. In the following, m refers to the number of clauses in F'.

Now we construct a Hopfield network in such a way that there is a stable vector $\tilde{u}$ corresponding to the truth assignment $\tilde{x}$, and unless there is a satisfying truth assignment, all input vectors differing from $\tilde{u}$ in at most k elements converge to $\tilde{u}$. On the other hand, if there is a satisfying truth assignment $\hat{x}$, the vector corresponding to $\hat{x}$ differs from the stable vector $\tilde{u}$ in at most k elements and does not converge to $\tilde{u}$; hence the attraction radius of $\tilde{u}$ is less than k.

[6] A CNF formula is a conjunction of clauses $c_1, c_2, \ldots, c_m$, where a clause is a disjunction of boolean variables $x_1, x_2, \ldots, x_k$ and their negations, e.g., $x_1 \vee \neg x_3 \vee x_4$; and a satisfying truth assignment is a choice of values (`true` or `false`) for the variables so that the formula gets value `true`.

In the construction, truth value `true` is represented by node state $+1$, and truth value `false` is represented by node state -1. In the following, we make no distinction between the truth values and the corresponding node states.

The network has nodes corresponding to the variables and the clauses, and $2k+2$ additional nodes, in total $3k+m+2$ nodes. We denote a state vector of the network as (x, c, r, s), where subvector $x = (x_1, \ldots, x_k)$ corresponds to the variables, subvector $c = (c_1, \ldots, c_m)$ corresponds to the clauses, and subvectors r and s, each of length $\beta = k+1$, correspond to the additional nodes.

Let c_x be the vector of truth values for the clauses resulting from assignment x. Especially, denote $\tilde{c} = c_{\tilde{x}}$. The stable vector in our construction will be $\tilde{u} = (\tilde{x}, \tilde{c}, -\overline{1}, -\overline{1})$.

Each r-node represents the conjunction of the clauses represented by the c-nodes; the r-nodes are replicated to guarantee that not all of them can be $+1$ for vectors within Hamming distance k from the stable vector $\tilde{u}$, which has $r = -\overline{1}$. The s-nodes work in such a way that as soon as *all* of them get state $+1$, their states cannot change any more.

Let $\alpha = \beta^2 + 1 = k^2 + 2k + 2$. The Hopfield network is constructed in the following way (see Figure 11):

- the threshold value of each node x_i is $-(\alpha m + 1)\tilde{x}_i$,
- the threshold value of each node c_j is $-\alpha(k_j - 1)$, where k_j is the number of literals (i.e., variables and negations of variables) in the clause c_j,
- the threshold value of each node r_i is $\beta(m-1)$,
- the threshold value of each node s_i is 0,
- there is an edge between each x_i and each c_j with weight α if literal x_i is in clause c_j, and with weight $-\alpha$ if literal $\neg x_i$ is in clause c_j,
- there is an edge between each c_j and each r_i with weight β,
- there is an edge between each r_i and the corresponding s_i with weight k,
- the s-nodes constitute a fully connected subnetwork: there is an edge between each s_i and each s_j (where $j \neq i$) with weight 1, and

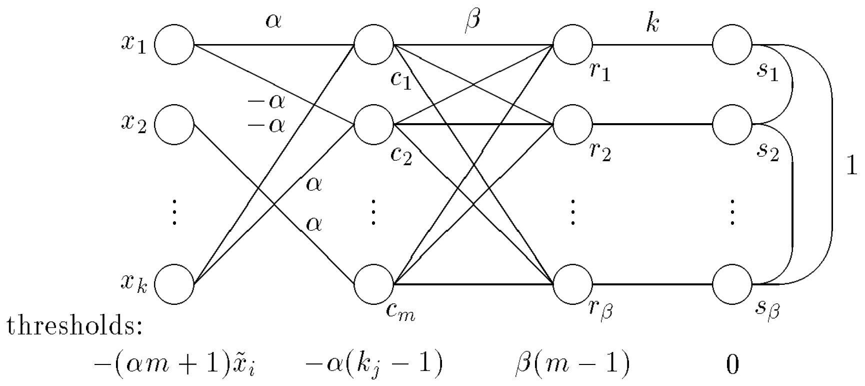

Figure 11: The structure of the Hopfield network in Theorem 5.10.

- all other edges have weight 0, i.e., they are excluded.

It is easy to check that $\tilde{u} = (\tilde{x}, \tilde{c}, -\overline{1}, -\overline{1})$ is a stable vector in this network.

This construction is polynomial in the size of the input formula, and we can now proceed to proving that this is the desired reduction. We prove first that if there is no satisfying truth assignment, all vectors at distance at most k from $\tilde{u}$ converge to $\tilde{u}$, and after that we prove that if there is a satisfying truth assignment $\hat{x}$, then vector $(\hat{x}, \tilde{c}, -\overline{1}, -\overline{1})$ does not converge to $\tilde{u}$, and hence the attraction radius of $\tilde{u}$ is strictly less than the Hamming distance between $\hat{x}$ and $\tilde{x}$, which is at most k.

1. Assume there is no satisfying truth assignment. In this case, take an arbitrary input vector (x, c, r, s) with Hamming distance at most k from $\tilde{u}$.

   At the first update step, the states of the x-nodes become $\tilde{x}$, since the input to node x_i is between $-\alpha m$ and αm, and hence x_i gets state $\operatorname{sgn}(\pm\alpha m + (\alpha m + 1)\tilde{x}_i) = \tilde{x}_i$. We use here an abbreviation of type $\operatorname{sgn}(a \pm b)$ for $\operatorname{sgn}(x)$, where $a - b \leq x \leq a + b$. As the threshold values in this way force the x-nodes to get states $\tilde{x}$, the states of the x-nodes do not change any more during the computation.

   The states of the c-nodes get values c_x. As there is no satisfying truth assignment, at least one of the c_x-values is -1.

   Recall that at least one c_j has initial state -1 even if k c-nodes have their states differing from $\tilde{c}$. Hence the input to each r-node from the c-nodes is at most $\beta(m-2)$, and as there is only one connection from an s-node to each r-node, the s-nodes contribute at most k to the input. Thus the r-nodes get state -1, and the only situation in which the r-nodes can get states 1 is when all the c-nodes have state 1.

At the second update step the states of the c-nodes become $\tilde{c}$. This can be seen as follows. As $\tilde{c}$ represents the truth values resulting from $\tilde{x}$, the input from the x-nodes is $-\alpha k_j$ if $\tilde{c}_j = -1$, and at least $-\alpha(k_j - 2)$ if $\tilde{c}_j = 1$. The input from the r-nodes is between $-\beta^2$ and β^2, so if $\tilde{c}_j = -1$, then c_j gets state $\operatorname{sgn}(-\alpha k_j \pm \beta^2 + \alpha(k_j - 1)) = -1$, and if $\tilde{c}_j = 1$, then c_j gets state $\operatorname{sgn}(-\alpha(k_j - 2) \pm \beta^2 + \alpha(k_j - 1)) = 1$. As the x-nodes do not change any more, also the c-nodes do not change any more during the computation.

As there is some c-node with state -1, the states of the r-nodes do not change: They are all still -1.

As the Hamming distance between (x, c, r, s) and $\tilde{u}$ is at most k, at least one r-node and at least one s-node have initial states -1. Thus at the first update step, the absolute value of the input from other s-nodes to each s-node is at most $k - 2$, whereas the input from the corresponding r-node is the state of the r-node times k. This results in the s-nodes having the same states after the first update step as the r-nodes had before the first update step. Consequently, there is at least one s-node with state -1 after the first update step. The first update step results in all r-nodes having state -1. Consequently, all s-nodes get states -1 in the second update step.

To sum up: Starting with (x, c, r, s), the first update step results in $(\tilde{x}, c_x, -\overline{1}, r)$, and the second update step results in $\tilde{u}$.

2. Assume that there is a satisfying assignment $\hat{x}$. We show that the input vector $(\hat{x}, \tilde{c}, -\overline{1}, -\overline{1})$ does not converge to $\tilde{u}$, which implies that the attraction radius must be less than k.

   In the first update step, the x-nodes become $\tilde{x}$, each c_j gets state $\operatorname{sgn}(-\alpha(k_j - 2) \pm \beta^2 + \alpha(k_j - 1)) = 1$, and r and s stay $-\overline{1}$. In the second update step, the c_j-nodes become $\tilde{c}$, but each r_i gets state $\operatorname{sgn}(\beta m \pm k - \beta(m-1)) = 1$, while s still stays $-\overline{1}$. In the third update step, the r-nodes become $-\overline{1}$ but each s_i gets state $\operatorname{sgn}(k - k) = 1$.

Now s stays as it is, since from now on the total input to each s_i is $-k+k=0$. The computation has converged to $(\tilde{x}, \tilde{c}, -\overline{1}, \overline{1}) \neq \tilde{u}$.

The proof is now complete. $\square$

From the construction in the proof, we see that just determining the two-step attraction radius is $\mathcal{NP}$-hard. Computing the direct attraction radius is easy while computing the two-step attraction radius is hard, because for the direct radius it is enough to check the change of one element at a time while for the two-step radius we have to check the changes of the changes.

Also approximating the attraction radius is hard. We say that an approximation algorithm to a minimization problem approximates the problem *within a factor* K, if for all sufficiently large problem instances I, the result of the algorithm is at most $K\min(I)$, where $\min(I)$ is the optimal result for instance I.

If a CNF formula is satisfiable, it can in general be satisfied by many different truth assignments. We use the name MIN ONES for the problem of finding the minimum number of `true` variables in a satisfying truth assignment.

We see from the construction of the network in Theorem 5.10 that the attraction radius is one less than the minimum Hamming distance between vector $\tilde{x}$ and a satisfying vector $\hat{x}$. Now, construct from a given instance of SAT a formula in the way described in Theorem 5.10. For each $\tilde{x}_i = 1$, change all literals x_i to $\neg x_i$ and all literals $\neg x_i$ to x_i. Now setting all variables to `false` yields a nonsatisfying truth assignment for the formula, and $(-\overline{1}, c_{-\overline{1}}, -\overline{1}, -\overline{1})$ is the stable vector we consider. Thus, the problem of computing the attraction radius is equivalent to the problem MIN ONES of finding the minimum number of `true` variables in a satisfying truth assignment to a CNF formula.

The MIN IND DOM SET problem asks for the size of a minimum independent dominating set in an undirected graph. Let (V, E) be a graph, where V is the set of nodes and $E \subseteq V \times V$ is the set of edges. Then a subset S of V is an independent dominating set if there is no edge between any two nodes in S and every node in the graph is either in S or is a neighbor of a node in S. Halldórsson has shown that MIN IND DOM SET cannot be approximated in polynomial time within a factor $n^{1-\epsilon}$ for any fixed $\epsilon > 0$, unless $\mathcal{P} =$

$\mathcal{NP}$ [35]. Here n is the number of nodes in the graph.

Lemma 5.11 *[47] There is no polynomial time algorithm approximating MIN ONES within a factor $n^{1-\epsilon}$ for any fixed ϵ, where $0 < \epsilon \leq 1$, unless $\mathcal{P}= \mathcal{NP}$. Here n is the number of variables in the CNF formula.*

PROOF: We prove that a polynomial time algorithm approximating MIN ONES within a factor K would give a polynomial time algorithm approximating MIN IND DOM SET within a factor K. Consequently, MIN ONES is at least as hard to approximate as MIN IND DOM SET, and the claim follows from Halldórsson's result.

Let (V, E) be a graph with n nodes. Create one variable s_i for each node $s_i \in V$. Note that for simplicity we use the same notation for both the node and the variable. Now we transform the property that a node is in an independent dominating set to the property that the corresponding variable is `true`. Denote the set of neighbors of node s_i by $E_i = \{\, s_j \mid \{s_i, s_j\} \in E \,\}$. Construct the CNF formula

$$G = \bigwedge_{s_i \in V} \Big(s_i \vee \bigvee_{s_j \in E_i} s_j \Big) \wedge \bigwedge_{\{s_i, s_j\} \in E} (\neg s_i \vee \neg s_j).$$

But every satisfying truth assignment to this formula corresponds to an independent dominating set in the graph and the size of this set is equal to the number of `true` variables. This completes the proof. □

As the problem of computing the attraction radius is equivalent to the problem MIN ONES of finding the minimum number of `true` variables, the following result is immediate.

Corollary 5.12 *There is no polynomial time algorithm approximating the attraction radii in a fully parallel Hopfield network (with n nodes) within a factor $n^{1-\epsilon}$ for any fixed ϵ, where $0 < \epsilon \leq 1$, unless $P = NP$.*

The results above for fully parallel operation can be extended to sequential operation. In the sequential operation case, the results are valid for the general attraction radius only; the k-step attraction radius is not interesting.

We sketch below how the proof of Theorem 5.10 must be modified in order to apply for sequential Hopfield memories. The nonapproximability result then follows in the same manner as in Corollary 5.12.

Theorem 5.13 *The problem "Given a sequential Hopfield network, a stable vector u, and a distance k; is the attraction radius of u less than k?" is $\mathcal{NP}$-hard.*

PROOF: (Sketch) The problem in applying the proof of Theorem 5.10 to the sequential case lies in the free update order. To avoid this problem, we add for each clause c_j a subnetwork checking that c_j has the correct value for the current variables x; i.e., the subnetwork computes $c_j \equiv (x_{i_1} \vee x_{i_2} \vee \cdots \vee x_{i_{k_j}})$.[7] The results of the subnetworks are used by the r-nodes: Node r_j gets value 1 if and only if $c_j = 1$ and, additionally, the equivalence is satisfied.

In order to avoid cheating the equivalence test by choosing suitable initial values, the subnetworks are replicated so that there are $k = \beta - 1$ such subnetworks for each c_j. Node r_j gets value 1 if and only if all the subnetworks connected to it yield 1. Additionally, to avoid cheating by manipulating a small set of the x variables, the result of the equivalence test must be `false` for the stable state $\tilde{u}$. Hence, we extend the equivalence test to

$$(c_j \equiv (x_{i_1} \vee x_{i_2} \vee \cdots \vee x_{i_{k_j}})) \wedge (x \not\equiv \tilde{x}).$$

The subnetworks are put between the layer of c-nodes and the layer of r-nodes: Each subnetwork has connections from each x-node and from the corresponding c-node and the result of the subnetwork is used by the corresponding r-node. Thus node r_j is connected to node c_j and to all the $\beta - 1$ subnetworks connected to c_j; each connection has weight β. (See Figure 12.) Each subnetwork has $3k + 5$ nodes and depth 4; the weights must again be chosen so that nodes to the right cannot influence the result (nodes to the left determine solely the outcome of the update). This means also that the weight α must be increased to $\alpha' = k_{\max}(\beta^2 + 2\beta - 3) + \beta + 1$, where $k_{\max}$ is the maximum number of literals in a clause in the formula.

Now we can proceed roughly as in Theorem 5.10. □

[7] Note that some variables may appear negated in the disjunction. For simplicity of notation, we assume that c_j is of the expressed form.

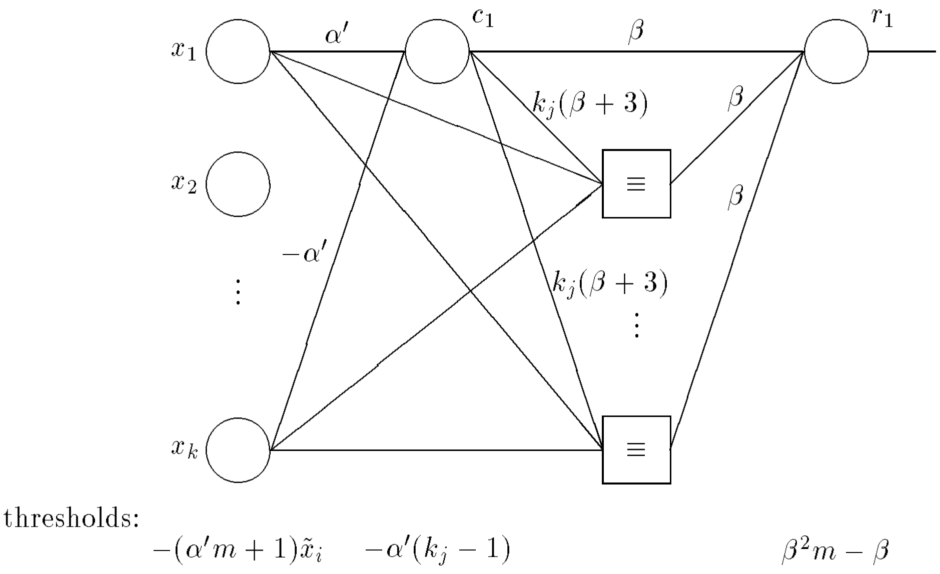

Figure 12: The network part with direct connections to node c_1 in the modified network in Theorem 5.13 (cf. Figure 11). Note that there in fact are four connections with different weights between each x-node and each subnetwork (denoted by a square).

5.5 Minimum Energy Computations

The use of a Hopfield network for optimization problems is based on the energy function analogy. The idea is to encode a problem instance into the weights of the network, in such a way that the energy function corresponds to the function that is to be optimized. Then the global minimum of the energy function corresponds to an optimal solution. We are usually satisfied with approximate solutions, i.e., local minima with energy values not far from the global minimum. The choice of initial vectors is a problem, as we are searching for the global minimum. In general, the initial vector must be near enough, in order to be able to produce the optimal solution. One approach is to choose randomly many different initial vectors and then perform the computation for them all. The best result is then given as the answer. However, Wiedermann [87, 88] has proved the following theorem, which we state without proof.

Theorem 5.14 *The problem "Given a sequential Hopfield network with small weights and an integer k; is there an initial vector of the network for which the final state vector has energy $\leq k$?" is $\mathcal{NP}$-complete.*

As it is a polynomial computation to reach the resulting vector from the initial vector when the weights are small, this theorem means that it is hard to choose initial vectors guaranteed to give good solutions to the optimization problem. Finding global optima in Hopfield networks is $\mathcal{NP}$-hard even when the nets are restricted to small nonpositive connection weights and small thresholds, since we can encode certain $\mathcal{NP}$-complete combinatorial optimization problems into Hopfield network structures [24, 60].

6 Conclusion

We have surveyed some aspects of the complexity of computations by discrete-time and discrete-state recurrent networks, specifically with symmetric connection weights. The most outstanding open problems in this area are possibly still those connected to the synthesis of Hopfield associative memories (section 4), i.e. developing easy-to-compute storage rules yielding high capacity and controlled attraction radii, and more generally determining the exact computational complexity of the synthesis problem (p. 26).

While the computational power of polynomial size Hopfield nets with hidden units was characterized in section 3 as equal to $\mathcal{PSPACE}$/poly, the power of nets with *no* hidden units (i.e., the basic Hopfield model) is still an open question. An interesting intermediate model in this respect might be the two-layer Hopfield net, or Bidirectional Associative Memory model of Kosko [57].

The results in Section 3, which were formulated for fully parallel operation, should be extended to cover also sequential operation. While sequentially updated nets are rather awkward to control (witness the difference between the Goles–Martínez and Haken constructions in Section 2.2), we do not expect any fundamental difficulties in this direction. Rather more interesting would be to develop analogous results for networks made up of unreliable elements; for some initial steps in this direction in the Siegelmann–Sontag network model, see the paper [78]. Eventually the theory should be extended also to computations by continuous-time networks; but here one is constrained by the current lack of generally accepted ways of measuring the complexity of continuous computations.

References

[1] N. Alon. Asynchronous threshold networks. *Graphs and Combinatorics,* 1:305–310, 1987.

[2] S.-I. Amari. Characteristics of sparsely encoded associative memory. *Neural Networks,* 2:451–457, 1989.

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

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

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

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

[7] J. L. Balcázar, J. Díaz, and J. Gabarró. On characterizations of the class $\mathcal{PSPACE}$/poly. *Theoretical Computer Science* 52:251–267, 1987

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

[9] E. B. Baum, J. Moody, and F. Wilczek. Internal representations for associative memory. *Biological Cybernetics,* 59:217–228, 1988.

[10] J. Bruck. On the convergence properties of the Hopfield model. *Proceedings of the IEEE,* 78:1579–1585, 1990.

[11] J. Bruck and J. W. Goodman. A generalized convergence theorem for neural networks. *IEEE Transactions on Information Theory,* 34:1089–1092, 1988.

[12] J. Bruck and J. Sanz. A study on neural networks. *International Journal of Intelligent Systems,* 3:59–75, 1988.

[13] V. Chandru, M. Vidyasagar, and V. Vinay. Theories for the synthesis of analog and discrete neural networks. Unpublished manuscript, 17 pp. July 1993.

[14] T. M. Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers,* 14:326–334, 1965.

[15] A. Dembo. On the capacity of associative memories with linear threshold functions. *IEEE Transactions on Information Theory,* 35:709–720, 1989.

[16] P. Floréen. Worst-case convergence times for Hopfield memories. *IEEE Transactions on Neural Networks,* 2:533–535, 1991.

[17] P. Floréen. *Computational Complexity Problems in Neural Associative Memories.* Doctoral dissertation, University of Helsinki, Department of Computer Science, Report A–1992–5, 1992.

[18] P. Floréen and P. Orponen. On the computational complexity of analyzing Hopfield nets. *Complex Systems,* 3:577–587, 1989.

[19] P. Floréen and P. Orponen. Attraction radii in Hopfield nets are hard to compute. *Neural Computation*, 5:812–821, 1993.

[20] F. Fogelman, E. Goles, and G. Weisbuch. Transient length in sequential iterations of threshold functions. *Discrete Applied Mathematics,* 6:95–98, 1983.

[21] S. Franklin and M. Garzon. Neural computability. In O. M. Omidvar, ed. *Progress in Neural Networks 1,* pp. 128–144. Ablex, Norwood, NJ, 1990.

[22] E. Gardner and B. Derrida. Optimal storage properties of neural network models. *Journal of Physics A: Math. Gen.,* 21:271–284, 1988.

[23] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness,* W. H. Freeman & Co. New York, NY, 1979.

[24] G. H. Godbeer. The computational complexity of the stable configuration problem for connectionist models. In *On the Computational Complexity of Finding Stable*

State Vectors in Connectionist Models (Hopfield Nets), Technical Report 208/88. University of Toronto, Department of Computer Science, 1988.

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

[26] E. Goles Ch. Dynamics of positive automata networks. *Theoretical Computer Science,* 41:19–32, 1985.

[27] E. Goles. Lyapunov functions associated to automata networks. In F. Fogelman Soulié, Y. Robert, and M. Tchuente, eds. *Automata networks in Computer Science — Theory and Applications,* pp. 58–81. Manchester University Press, 1987.

[28] E. Goles Ch., F. Fogelman Soulié and D. Pellegrin. Decreasing energy functions as a tool for studying threshold networks. *Discrete Applied Mathematics,* 12:261–277, 1985.

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

[30] E. Goles and S. Martínez. *Neural and Automata Networks: Dynamical Behavior and Applications,* Kluwer Academic Publishers, Dordrecht, 1990.

[31] E. Goles Ch. and J. Olivos A. The convergence of symmetric threshold automata. *Information and Control,* 51:98–104, 1981.

[32] A. Haken. Connectionist Networks that Need Exponential Time to Stabilize. Unpublished manuscript, Department of Computer Science, University of Toronto, 1989.

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

[34] M. Hall, Jr. *Combinatorial Theory, 2nd Ed.* John Wiley & Sons, New York, NY, 1986.

[35] M. M. Halldórsson. Approximating the minimum maximal independence number. *Information Processing Letters,* 46:169–172, 1993.

[36] R. Hartley and H. Szu. A comparison of the computational power of neural network models. *Proc. IEEE First International Conference on Neural Networks, Vol. III*, pp. 15–22. IEEE, New York, 1987.

[37] D. O. Hebb. *The Organization of Behavior.* John Wiley & Sons, New York, NY, 1949.

[38] J. Hertz, A. Krogh and R. G. Palmer. *Introduction to the Theory of Neural Computation.* Santa Fe Institute Studies in the Sciences of Complexity: Lecture Notes Vol. I. Addison-Wesley, Redwood City, CA, 1991.

[39] J. Hong. On connectionist models. *Comm. Pure and Applied Math.*, XLI:1039–1050, 1988.

[40] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, MA, 1979.

[41] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proc. National Academy of Sciences of the USA*, 79:2554–2558, 1982.

[42] J. J. Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *Proc. National Academy of Sciences of the USA*, 81:3088–3092, 1984.

[43] J. J. Hopfield and D. W. Tank. Neural computation of decisions in optimization problems. *Biological Cybernetics*, 52:141–152, 1985.

[44] *IEEE Transactions on Neural Networks*, 3 (1992) 3 (May). Special issue on neural network hardware.

[45] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37:79–100, 1988.

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

[47] V. Kann. Personal communication, 1992.

[48] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, eds. *Complexity of Computer Computations*, pp. 85–103. Plenum Press, New York, NY, 1972.

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

[50] R. M. Karp, E. Upfal, and A. Widgerson. Constructing a perfect matching is in Random $\mathcal{NC}$. *Combinatorica*, 6:35–48, 1986.

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

[52] T. Kohonen. Correlation matrix memories. *IEEE Transactions on Computers*, 21:353–359, 1972. Reprinted in [5], pp. 174–180.

[53] T. Kohonen. *Self-Organization and Associative Memory*. Springer-Verlag, Berlin, 1989 (3rd Edition).

[54] T. Kohonen, K. Ruohonen. Representation of associated data by matrix operations. *IEEE Transactions on Computers*, 22:701–702, 1973.

[55] J. Komlós. On the determinant of (0,1) matrices. *Studia Scientiarum Mathematicarum Hungarica*, 2:7–21, 1967.

[56] J. Komlós and R. Paturi. Convergence results in an associative memory model. *Neural Networks*, 1:239–250, 1988.

[57] B. Kosko. Bidirectional associative memories. *IEEE Transactions on Systems. Man, and Cybernetics*, 18:49–60, 1988. Reprinted in [6], pp. 165–176.

[58] A. Kuh and B. W. Dickinson. Information capacity of associative memories. *IEEE Transactions on Information Theory*, 35:59–68, 1989.

[59] M. Lepley and G. Miller. Computational Power for Networks of Threshold Devices in an Asynchronous Environment. Technical Report, Department of Mathematics, Massachusetts Instititute of Technology, 1983.

[60] J. Lipscomb. On the computational complexity of finding a connectionist model's stable state vectors. In *On the Computational Complexity of Finding Stable State Vectors in Connectionist Models (Hopfield Nets)*, Technical Report 208/88. University of Toronto, Department of Computer Science, 1988.

[61] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics,* 5:115–133, 1943. Reprinted in [5], pp. 18–27.

[62] R. J. McEliece, E. C. Posner, E. R. Rodemich, and S. S. Venkatesh. The capacity of the Hopfield associative memory. *IEEE Transactions on Information Theory,* 33:461–482, 1987.

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

[64] B. L. Montgomery and B. V. K. Vijaya Kumar. Evaluation of the use of the Hopfield neural network model as a nearest-neighbor algorithm. *Applied Optics,* 25:3759–3766, 1986.

[65] S. Muroga, I. Toda and S. Takasu. Theory of majority decision elements. *J. Franklin Inst.,* 271:376–418, 1961.

[66] K. Nakano. Associatron — a model of associative memory. *IEEE Transactions on Systems, Man, and Cybernetics,* 12:380–388, 1972. Reprinted in [6], pp. 90–98.

[67] P. Orponen. On the computational power of discrete Hopfield nets. In *Proc. 20th International Colloquium on Automata, Languages, and Programming,* pp. 215–226. Lecture Notes in Computer Science 700. Springer-Verlag, Berlin, 1993.

[68] P. Orponen. Computational complexity of neural networks: A survey. *Nordic Journal of Computing* 1 (1994), 94–110. (A preliminary version appeared as "Neural networks

and complexity theory" in *Proc. 17th International Symposium on Mathematical Foundations of Computer Science,* pp. 50–61. Lecture Notes in Computer Science 629. Springer-Verlag, Berlin, 1992.)

[69] I. Parberry. A primer on the complexity theory of neural networks. In R. B. Banerji, ed. *Formal Techniques in Artificial Intelligence: A Sourcebook,* pp. 217–268. Elsevier, Amsterdam, 1990.

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

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

[72] S. Poljak and M. Sůra. On periodical behaviour in societes with symmetric influences. *Combinatorica,* 3:119–121, 1983.

[73] S. Porat. Stability and looping in connectionist models with asymmetric weights. *Biological Cybernetics,* 60:335–344, 1989.

[74] P. Raghavan. Learning in threshold networks. In *Proc. of the 1988 Workshop on Computational Learning Theory,* pp. 19–27. Morgan Kaufmann, San Mateo, CA, 1988.

[75] E. Sánchez-Sinencio and C. Lau (eds.). *Artificial Neural Networks: Paradigms. Applications, and Hardware Implementations,* IEEE, New York, NY, 1992.

[76] A. A. Schäffer and M. Yannakakis. Simple local search problems that are hard to solve. *SIAM Journal on Computing,* 20:56–87, 1991.

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

[78] H. T. Siegelmann. On the computational power of probabilistic and faulty neural networks. In *Proc. 21st International Colloquium on Automata, Languages, and Programming,* pp. 23–34. Lecture Notes in Computer Science 820. Springer-Verlag, Berlin, 1994.

[79] H. T. Siegelmann, and E. D. Sontag. On the computational power of neural nets. In *Proc. 5th Annual Workshop on Computational Learning Theory,* pp. 440–449. Morgan Kaufmann, San Mateo, CA, 1992. (Full version to appear in *Journal of Computer and System Sciences.*)

[80] P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. In D. E. Rumelhart and J. L. McClelland, eds. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1,* pp. 194–281. The MIT Press, Cambridge, MA, 1986.

[81] F. Tanaka and S. F. Edwards. Analytic theory of the ground state properties of a spin glass: I. Ising spin glass. *Journal of Physics F: Metal Phys.,* 10:2769–2778, 1980.

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

[83] S. S. Venkatesh. Epsilon capacity in neural networks. In J. Denker, ed. *Proc. Neural Networks for Computing,* pp. 440–445. AIP Conference Proceedings Vol. 151. American Institute of Physics, New York, NY, 1986.

[84] S. S. Venkatesh and D. Psaltis. Linear and logarithmic capacities in associative neural networks. *IEEE Transactions on Information Theory,* 35:558–568, 1989.

[85] S. S. Venkatesh and D. Psaltis. On reliable computation with formal neurons. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 14:87–91, 1992.

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

[87] J. Wiedermann. Complexity issues in discrete neurocomputing. In *Proc. Aspects and Prospects of Theoretical Computer Science,* pp. 480–491. Lecture Notes in Computer Science 464. Springer-Verlag, Berlin, 1990.

[88] J. Wiedermann. On the computational efficiency of symmetric neural networks. *Theoretical Computer Science,* 80:337–345, 1991.

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