Attraction radii in binary Hopfield nets are hard to compute

Patrik Floréen*

Pekka Orponen

Department of Computer Science, University of Helsinki

SF–00510 Helsinki, Finland

Abstract

We prove that it is an *NP*-hard problem to determine the attraction radius of a stable vector in a binary Hopfield memory network, and even that the attraction radius is hard to approximate. Under synchronous updating, the problems are already NP-hard for two-step attraction radii; direct (one-step) attraction radii can be computed in polynomial time.

A *Hopfield memory network* [6] consists of n binary valued nodes, or "neurons." We index the nodes by $\{1, \ldots, n\}$, and choose $\{-1, +1\}$ as their possible states (the values $\{0, 1\}$ could be chosen equally well). Associated to each pair of nodes i, j is an *interconnection weight* w_{ij}. The interconnections are symmetric, so that $w_{ij} = w_{ji}$ for each i, j; moreover, $w_{ii} = 0$ for each i. In addition, each node i has an internal *threshold value* t_i. We denote the matrix of interconnection weights by $W = (w_{ij})$, and the vector of threshold values by $t = (t_1, t_2, \ldots, t_n)$.

At any given moment, each node i in the network has a state x_i, which is either -1 or $+1$. The state at the next moment is determined as a function of the states of the other nodes as

*Work supported by the Academy of Finland.

$x_i := \text{sgn}(\sum_{j=1}^{n} w_{ij} x_j - t_i)$, where sgn is the signum function ($\text{sgn}(x) = 1$ for $x \geq 0$ and $\text{sgn}(x) = -1$ for $x < 0$).

In the *synchronous* network model, this update step is performed simultaneously for all the nodes. Thus we may denote the global update rule for the network as $x := \text{sgn}(Wx - t)$, where $x = (x_1, x_2, \ldots, x_n)$ is the vector of states for the nodes. It is known [4] 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 $\text{sgn}(Wu - t) = u$), or to a cycle of length two (i.e., vectors $u \neq v$ such that $\text{sgn}(Wu - t) = v$ and $\text{sgn}(Wv - t) = u$). We are only interested in the stable solutions.

In an *asynchronous* network, the update step is performed for one node at a time in some (usually random) order. As shown in [6], in a network with $w_{ii} = 0$ for each i, all asynchronous computations eventually converge to stable vectors.

The *attraction radius* of a stable vector u is the largest Hamming distance from within which all other vectors are guaranteed eventually to converge to u. The *k-step* attraction radius is the largest distance from within which all other vectors converge to u in at most k update steps. As the main interest in Hopfield nets is in their error-correcting capacity with respect to the stable vectors, it would be of considerable importance to be able to determine the attraction radius of a given stable vector. However, we show that this problem is NP-hard for both the synchronous and the asynchronous networks, and thus not solvable by a polynomial time algorithm unless $P = NP$. Even more, we show that the attraction radius of a given stable vector (of length n) cannot be even *approximated* in polynomial time within a factor $n^{1-\epsilon}$ for any fixed $\epsilon > 0$, unless $P = NP$. (Similar results for other problems arising in the context of analyzing Hopfield nets have been obtained in [1, 3]. For general introductions to computational complexity issues in neural networks, see [8, 10, 11].)

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

Theorem 1 *The problem "Given a synchronous 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 x:

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 $x_i = 1$ 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 $x_i = -1$ 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 x. If this distance is k, the radius of direct attraction is $k - 1$. □

Next we consider the problem of computing the asymptotic attraction radius. Note that this problem is in *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 [4].

Theorem 2 *The problem "Given a synchronous Hopfield network, a stable vector u, and a distance k; is the attraction radius of u less than k?" is NP-hard.*

Proof We prove that the problem is NP-hard by a reduction from the 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?"[1] 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 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

[1] 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`.

elements and does not converge to $\tilde{u}$; hence the attraction radius of $\tilde{u}$ is less than k.

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 1):

- 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,

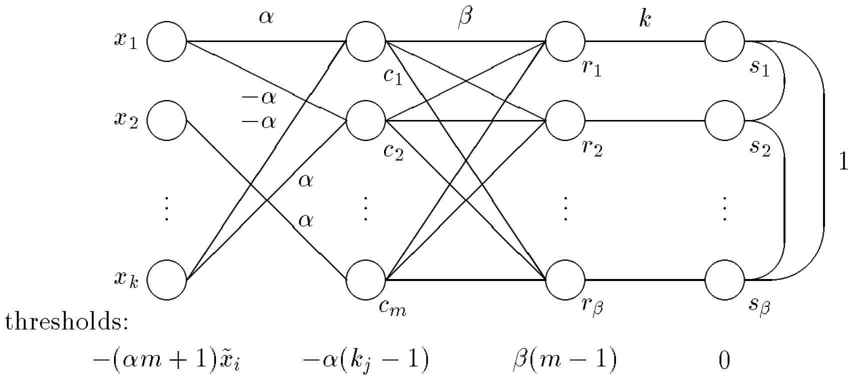

Figure 1: The structure of the Hopfield network in Theorem 2.

- 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
- 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 $k - 3$, 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 $\mathrm{sgn}(-\alpha(k_j - 2) \pm \beta^2 + \alpha(k_j - 1)) = 1$, 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 $\mathrm{sgn}(\beta m \pm k - \beta(m-1)) = 1$, while s still stays $-\overline{1}$. In the third update step, the r-nodes become $-\overline{1}$ but each s_i gets state $\mathrm{sgn}(k - k) = 1$. Now s stays as this, as 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 completed. □

From the construction in the proof, we see that already determining the two-step attraction radius is 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 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. (The analogous maximization problem Max Ones has been considered in [9].)

We see from the construction of the network in Theorem 2 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 2. 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.

It is easy to show that there is no polynomial time algorithm approximating Min Ones within a factor K for any fixed $K > 1$, unless $P = NP$. Given a CNF formula F with k variables, denote $n = \lfloor Kk \rfloor$ and add $n + 1$ new variables z, z_1, z_2, $\ldots$, z_n. Construct the formula

$$G = (F \vee z) \wedge \bigwedge_{i=1}^{n} [(z \vee \neg z_i) \wedge (\neg z \vee z_i)].$$

Note that G can be made into a CNF formula by distributing the z in the first conjunct over the clauses of F. Now the number of `true` variables needed to make G `true` is either at most k (if F is satisfiable) or $n + 1 > Kk$ (setting z_1, z_2, $\ldots$, z_n, and z to `true`). Consequently, an algorithm approximating Min Ones within a factor K would in fact decide the satisfiability of formula F.

We shall introduce here also a stronger construction, which was suggested to us by Viggo Kann [7]. For this construction, we need the Minimum Independent Dominating Set minimization problem, which 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. Magnús Halldórsson has shown that Minimum Independent Dominating Set cannot be approximated in polynomial time within a factor $n^{1-\epsilon}$ for any fixed $\epsilon > 0$, unless $P = NP$ [5]. Here n is the number of nodes in the graph.

Lemma 1 *There is no polynomial time algorithm approximating* Min Ones *within a factor $n^{1-\epsilon}$ for any fixed $\epsilon > 0$, unless $P = NP$. 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 Minimum Independent Dominating Set within a factor K. Consequently, Min Ones is at least as hard to approximate as Minimum Independent Dominating 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. $\square$

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

Proof We have already seen that the problem of computing the attraction radius is equivalent to the problem Min Ones of finding the minimum number of `true` variables. The claim follows immediately from Lemma 1. □

The results above for synchronous Hopfield memories can be extended to asynchronous Hopfield memories. In the asynchronous case, the results are valid for the asymptotic attraction radius only; the k-step attraction radius is not interesting.

We sketch below how the proof of Theorem 2 must be modified in order to apply for asynchronous Hopfield memories. The nonapproximability result then follows in the same manner as in Corollary 1.

Theorem 3 *The problem "Given an asynchronous Hopfield network, a stable vector u, and a distance k; is the attraction radius of u less than k?" is NP-hard.*

Proof (sketch) The problem in applying the proof of Theorem 2 to the asynchronous 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 $c_j \equiv (x_{i_1} \vee x_{i_2} \vee \cdots \vee x_{i_{k_j}})$.[2] 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

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

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

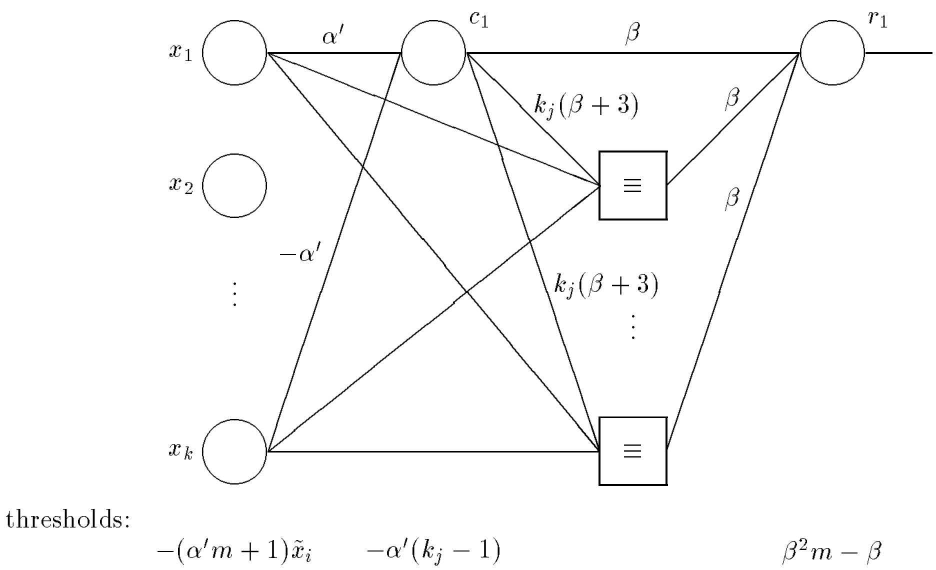

Figure 2: The network part with direct connections to node c_1 in the modified network in Theorem 3 (cf. Figure 1). Note that there in fact are four connections with different weights between each x-node and each subnetwork (denoted by a square).

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 2.) 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 2. □

Acknowledgments

The second author would like to thank Max Garzon for inspiring discussions on the topics of this work.

References

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

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

[3] G. H. Godbeer, J. Lipscomb, and M. Luby, *On the Computational Complexity of Finding Stable State Vectors in Connectionist Models (Hopfield Nets).* Technical Report 208/88, Dept. of Computer Science, Univ. of Toronto, 1988.

[4] E. Goles Ch., F. Fogelman-Soulie, and D. Pellegrin, "Decreasing energy functions as a tool for studying threshold networks," *Discrete Applied Mathematics* 12 (1985), 261–277.

[5] M. M. Halldórsson, "Approximating the minimum maximal independence number," submitted for publication, 1992.

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

[7] V. Kann, personal communication, 1992.

[8] P. Orponen, "Neural networks and complexity theory," in *Proc. 17th International Symposium on Mathematical Foundations of Computer Science.* Lecture Notes in Computer Science 629, Springer-Verlag, Berlin, Germany, 1992, 50–61.

[9] A. Panconesi and D. Ranjan, "Quantifiers and approximation," in *Proc. 22nd Annual ACM Symposium on Theory of Computing.* ACM, New York, 1990, 446–456.

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

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