Computational Complexity Problems in Neural Associative Memories

Patrik Floréen

University of Helsinki, Department of Computer Science

Teollisuuskatu 23, SF-00510 Helsinki, Finland

Contents

1 Introduction

Since the work by Cajal [20] in the beginning of this century, it has been known that the brain consists of a very large number of neurons. There are 10^{10} to 10^{11} neurons in the human nervous system [105, p. 241]. Despite the slow operation of the individual neurons, the brain can efficiently handle computationally complicated tasks, such as image processing. Moreover, it is fault tolerant, can deal with incomplete and inconsistent information, can learn from experience, and is small and consumes little power.

Inspired by these facts, *artificial neural networks* have been constructed. In such a network, the computational power stems from the collective simultaneous work of very many simple processors (*massive parallelism*), and the information needed in the computation is encoded in the connections between the processors (*connectionism*). After the information encoding is completed, the computations in the network are performed by the processors repeatedly executing very simple operations.

In the 1940s, McCulloch and Pitts modeled a neuron by a binary threshold unit [85] and Hebb proposed a theory of learning [60], while at the same time von Neumann sketched the main lines of the computer architecture in use today [94]. The von Neumann architecture proved successful and by the end of the 1950s neural network research stagnated. In the 1960s Rosenblatt's perceptron model [103, 104] became very popular until Minsky and Papert [88] investigated the limitations of this model and made these limitations widely known. Not until the 1980s did the interest revive. The time was ripe: Implementing artificial neural networks in hardware was becoming possible. Also theoretical breakthroughs, (e.g., the backpropagation learning algorithm [106], originally introduced by Werbos [124]) and the example given by influential scientists starting to investigate the field (e.g., Hopfield [64]) had strong impact.

Theoretical work has provided a firm ground for further development and the research in neural networks is now pursued at a steady pace, but still on a broad front. In Japan, a new "sixth generation" computer project has just started in April 1992. This 10-year program with the official name *Real-World Computing (RWC)* has a strong emphasis on neural networks [69]. Today there are many neural network applications, see for instance [108]. Links to other disciplines are being discovered. The area is maturing and becoming part of everyday computer science. Even in beginner's books on ADP, neural networks are discussed briefly (e.g., [110]).

The field of neural networks draws attention from as different parts of the scientific community as mathematics, physics, computer science, engineering, psychology, medicine, and biology. While the biologists aim at explaining the structure and operation of the human brain mathematically, the computer science point of view, which is adopted in this thesis, is different; the argument of biological plausibility is not used, and biology provides at most some inspiration for the research. Still, we will use some biologically inspired terminology, since it is customary in this area, but these concepts must be understood to be without actual biological relevance. Also, real neurons have very low precision, while we assume here high precision as in traditional computers. In the rest of this text, we refer to artificial neural networks by the name neural networks.

There are several reasons for studying neural networks. Firstly, there is hope for implementing the massively parallel computations in efficient hardware. Semiconductor gates are about one million times faster than biological neurons! The vast number of connections is a problem, where optical computers can be a solution. See [67, 108] for collections of papers about hardware implementations of neural networks. Secondly, neural networks are supposed to generalize in a "sensible" way. After given some example instances, also new instances not formerly encountered can be handled. Thirdly, the scenario of networks learning instead of being programmed opens up exciting possibilities. Learning, which is a dominant theme in the neural network research today, is however not discussed in this thesis.

Neural networks provide as computational structures an alternative to the traditional von Neumann computer architecture. The notion of computability is nevertheless unchanged. All problems computable on an artificial neural network with a finite number of neurons and finite number of possible states for each neuron are computable on a finite automaton, and vice versa [58, 85, 87]. For more discussion about this subject, see the survey by Orponen [96].

Biological memory is *associative*: We can retrieve information when provided with only a noisy part of the key to it. Almost every application of neural networks can be thought of as association in some way; also generalization is a form of association. Artificial neural associative memories were first studied by Taylor [115]. Associative memories attracted much attention from the mid-1950s to the mid-1970s, but because no significant practical applications emerged, the interest declined. The general interest awoke again after Hopfield's study [64] of the memory model that now bears his name.

Associative memories can perform error correction and pattern recognition. They can be applied to problems in coding theory; decoding a binary error-correcting code used on a noisy memoryless binary symmetric channel is in fact equivalent to the computation performed by an associative memory. Associative memories are useful is data compression, where the source data are mapped to a set of transmittable code words. Recognition and reconstruction of images, and retrieval of bibliographic information from partial references are mentioned in [62] as examples of applications for associative memories. In [25] associative memories are suggested for searching large databases, large sets of rules in expert systems, or objects with similar attributes in object descriptions. In [90] the possibility of using associative memory structures for cache memories in hierarchical memory systems is put forward. In [93] an autonomous moving robot uses an associative memory for its image processing.

When evaluating the usefulness of a specific neural associative memory model, we have to keep in mind at least the following aspects: quality of retrieved results, retrieval time, fault tolerance, and hardware implementability. Our main concerns here are the first two aspects: the correctness and the speed of the memory. Fault tolerance can be achieved by redundancy [95]; simplicity of the model enhances the hardware implementability. Both of the main associative memory models studied in this thesis, the *Hopfield* and the *Hamming* associative memories, have been implemented in hardware [29, 70, 102, 107].

Excellent overviews of different aspects of neural networks are provided by [8, 52, 61, 62, 71, 77]. Amit [8] gives a pure physicist's point of view of mainly Hopfield networks;

Goles and Martínez [52] describe automata networks and the use of Lyapunov functions for their study; Hecht-Nielsen's book [61] is a survey of different neural network architectures and learning algorithms; Hertz, Krogh, and Palmer [62] offer a physicist's point of view of many neural network models; the book by Kamp and Hasler [71] is an excellent monograph covering theoretical results for associative memories; and Kohonen's book [77] includes the correlation matrix memory and the theory of learning vector quantization and feature maps. General introductions to computational complexity issues in neural networks are provided by [98, 125] and more recently by [96]. For an overview of the history of neural network research, see [24].

In this thesis, which expands [32], we explore theoretical issues about associative memories realized as neural networks. The works [14, 40, 49, 82] have been the main sources of inspiration for this research. Of the books mentioned above, Kamp's and Hasler's [71] has been the most important one for our work. In that book, the problems in the study of neural associative memories are divided into analysis and synthesis problems and the different approaches to these problems into deterministic, statistical, and thermodynamical. The analysis problems concern properties of given networks; the synthesis problems are about methods to produce networks with given properties. In contrast to the deterministic approach, the statistical and the thermodynamical approaches rely on hypotheses about the statistical distribution of some property. Average case analyses are typical examples of the statistical approach. In the thermodynamical extension, the probability distribution employed depends on a parameter called *temperature*, which is lowered towards zero during the computation. In Kamp's and Hasler's terminology, our study is concerned with *analysis* problems from a *deterministic* point of view.

We restrict our study to the Hopfield and the Hamming memories (with some extensions) with *discrete* valued input, output, and stored vectors. The Hopfield memory model has attracted much attention, partly because of its intriguing dynamics, and partly because of its analogy with certain well-studied physical systems (Ising spins in the theory of magnetism). However, the capacity and reliability of the Hopfield memory have proved to be unsatisfactory. The other model, the Hamming memory, is faster and more reliable, and makes more efficient use of its resources. Related to these models are the sparse distributed memories, such as Kanerva's memory [72], but they are not considered here. There are several more complex structures to be found in the literature, but the models studied here can be thought of as basic components of larger systems. It is fundamental to understand the behavior of these simple models.

There are three main themes in this thesis:

1. the convergence of computations, including methods to speed up the convergence,
2. the computational complexity of assessing the quality of a given Hopfield memory network, and
3. the effect of "don't know" elements in the input and the output.

We also report on simulation experiments that illustrate the questions covered by our theoretical studies.

The organization of this thesis is as follows. Chapter 2 gives the basic terminology needed in the rest of the thesis. Chapter 3 studies the basic properties of Hopfield memories. The capacity of Hopfield networks has been well studied, so we only give a brief summary of that problem. However, we show that the convergence time under asynchronous updating is at most about $\frac{1}{2}\sum_{i,j}|w_{ij}|$, where the values w_{ij} are the elements of the connection matrix. This bound is about half of the best previously known upper bound [52]. When "don't know" elements are allowed in the input and the output, the upper bound we obtain on the convergence time under asynchronous updating is about $2\sum_{i,j}|w_{ij}|$. For synchronous updating, we obtain corresponding results, thus expanding the results of Goles [49] to more general cases.

Chapter 4 concerns the computational complexity of evaluating the quality of a given Hopfield memory. All the essential quality questions posed (number of stable vectors, sizes of attraction domains, attraction radii) are shown to be *NP*-hard. The complexity of determining the number of stable vectors has also been studied by Lipscomb et al. [46].

Chapter 5 presents the Hamming memory. While the capacity of a Hamming network is a trivial issue, this is not the case for the convergence. In [82] the network is shown to converge when there is a unique nearest stored vector to the input vector. We show that the probability for random vectors to have such a unique nearest stored vector approaches 1 as the vector length grows, if the number of stored vectors grows subexponentially in the vector length. However, situations with several nearest stored vectors are rather frequent in practice. We study which conditions the parameters of the network must satisfy in order to ensure convergence in all possible cases. The actual convergence time has not been analysed before, but we obtain a tight bound on the convergence time and show that with an optimal choice of parameter values the worst-case convergence time is $\Theta(p \log(pn))$, where p is the memory capacity (i.e., the maximum number of stored vectors) and n is the vector length. Restricting the instances to those with a unique nearest stored vector to the input vector, the worst-case convergence time is $\Theta(p \log n)$.

After this, we consider some modifications to the structure of the Hamming memory. First, methods to speed up the convergence are developed, and then, in Chapter 6, the Hamming memory is modified into a new memory model with a feed-forward network structure. As there is always a trade-off between the number of computation steps and the complexity of each step, the price we have to pay for these modifications is more complex local computations. In simulations, the behavior of the new memory model is demonstrated.

Chapter 7 finally gives a summary, some concluding remarks, and some suggestions for further research.

Parts of the research reported in this thesis have been or will be presented in separate papers [31, 33, 34, 35, 36, 37, 38, 39].

2 Neural Networks as Associative Memories

This chapter introduces our terminology on neural networks and associative memories. It also gives motivation for the use of "don't know" elements. For the mathematical conventions used in the thesis, see Appendix I.

2.1 Neural Networks

We first describe the structure of our neural networks and then turn to the computations performed. A neural network consists of processing elements called *nodes* or *neurons*, and real-valued weighted arcs, *connections*, between them. As is customary in the neural networks literature, we denote the weight on the connection to node i from node j by w_{ij}. The connections are often called synaptic interconnections, and the weights w_{ij} are called synaptic weights. The matrix of weight values, the *connection matrix* $W = (w_{ij})$ is, analogously, often called the synaptic matrix. As we are not trying to give models for real neurons in the brain, we prefer to talk about nodes and connections. If the connections are directed and there are no cycles or loops in the network, we have a *feed-forward* network. In this case we can partition the neurons into distinct groups, so that all connections from the neurons in group i go to neurons in groups j, for $j > i$. We call such a group a *layer*. We will also occasionally use the word layer for a distinct group of nodes, even if the network is not strictly feed-forward.

Every node i has an internal *state* x_i, which is also the output from the node. This state is usually binary, but can be even a real value. The collection of states of all the nodes can be interpreted as elements of a vector $x = (x_1, x_2, \ldots, x_n)$, expressing the state of the network.[1] Each node i has also an individual *threshold* value θ_i and a *node function* f_i. We will allow rather complicated node functions in Section 5.4 and Chapter 6. The node function is part of the network architecture, and is thus not changed during the computations. The node function is usually identical for all nodes or within each layer.

[1] In fact, the vector is $x = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}$. Generally, we will not care about this detail, as it is always clear from the context what is intended.

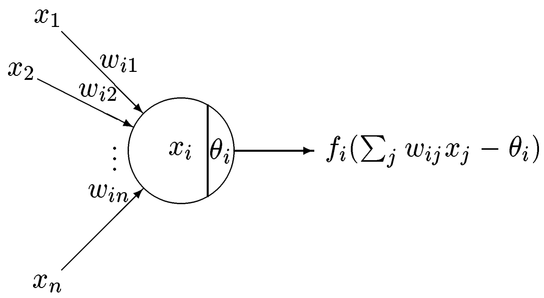

Figure 1: A McCulloch-Pitts formal neuron.

The threshold is sometimes called bias, and the node function is called transfer function or transition function.

There are two types of computations. The "programming" of the network is performed by setting the connection weights and the threshold values of the neurons. The network is then used in the following way. The input to the network initializes the states of a set of nodes to the values of the elements of the input vector. These nodes corresponding to the input vector are the *input nodes*. Analogously, the output vector can be read out as the final states of certain nodes, the *output nodes*. The actual computations are performed by each node locally updating its state: The weighted sum of the states of the nodes connected to it is decreased by the threshold value, and the node function is applied to the result. In other words, the state x_i is updated to $f_i(\sum_j w_{ij}x_j - \theta_i)$. The connection weights and the threshold values remain unchanged during this computation. A node of the described type, often called a McCulloch-Pitts formal neuron, is shown in Figure 1. All nodes discussed in this thesis are McCulloch-Pitts neurons.

The thresholds θ_i can always be substituted for by an additional node with fixed state 1 and, for all nodes i, a connection with weight $-\theta_i$ from it to node i.

If there is feedback, the computation is iterative: The nodes update their states repeatedly. The computation is terminated when *convergence* is reached, meaning that no states can any longer be changed. The nodes are updated either *synchronously* or *asynchronously*: in synchronous updating all nodes are updated simultaneously in parallel, in asynchronous updating the nodes are updated one at a time in some order.

The order in which the nodes are updated in asynchronous updating is usually a *fair random* order, but can also be a fixed sequential order. Fairness means here that at any point in the computation, any node would be updated in the future if the computation were executed forever. From a practical point of view, it is enough to require that any node is updated at least once during any period of some given length T. In this case we know that the computation has converged when there have been no state changes during the last T iterations. We can also have *partially synchronous* updating, as in the Hamming memory. In such updating, the nodes within certain subsets are updated synchronously, and the different subsets are updated in some specific order.

To sum up: The network architecture is determined by a set of nodes with their node functions, and a set of arcs between the nodes. Given a fixed architecture, the network is "programmed" by setting the arc weights and the threshold values of the nodes. Given such a fixed setting, a computation is performed by initializing the states of the nodes and then repeatedly updating the nodes. After convergence is reached, the result is read out from the resulting states of the nodes.

2.2 Associative Memories

There are many different definitions of content addressable memory (CAM) and associative memory. In [14] we find the following definitions: a CAM "allows recall of a stored word without reference to the physical location where the word is stored", and an associative memory "allows recall based on only partial, or partially erroneous information". The definitions in [25, 77] are equivalent to this. In [127] an associative memory is a storage device that stores pairs of patterns (u, v) in such a way that presentation of u will bring v from store. All the memory models we discuss in this thesis are both associative and content addressable according to these definitions. Many authors talk about associative content addressable memories, or use either of the names for them. In the following we will use the term "associative memory" and make the following definition.

An *associative memory* is a storage device that stores pairs of vectors (u^i, v^i), for $i = 1, 2, \dots, m$, 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 as result an *output vector* close to v^i. The memory is *autoassociative* if $u^i = v^i$ for all i; otherwise it is *heteroassociative*.

The stored vectors are often called prototypes or stored patterns; the input vectors are called test patterns. Viewing the pairs (u^i, v^i) as defining part of a mapping φ, with $\varphi(u^i) = v^i$ for $i = 1, 2, \dots, m$, the associative memory maps x to the estimate $\varphi(x) \approx \varphi(u^i)$, when $x \approx u^i$.

Observe that there are always two types of computations in an associative memory: the storage of the patterns and the retrieval from store.

While in error correction applications, the Hamming distance (i.e., the number of differing bits) is the correct similarity metric, this is not the case in, for instance, image processing, where we need translation invariance. This problem can be overcome by preprocessing. In [93], an autonomous moving robot handles the image processing with a Hamming associative memory (though the name Hamming memory is not mentioned in the paper). Consequently, we assume the metric to be the Hamming metric or some usual p-metric (see Appendix I).

A memory is *distributed*, if the storage of the memory elements is realized in such

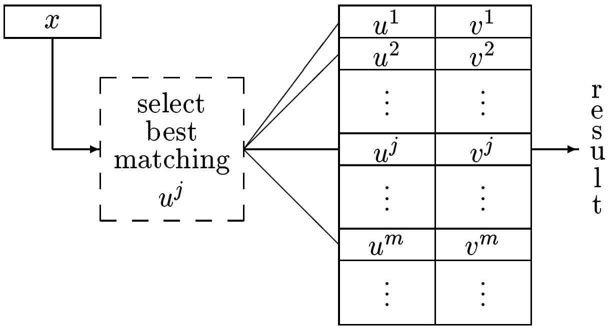

Figure 2: The trivial associative memory.

a way that each stored element (or different parts thereof) is stored in many memory locations, and each memory location holds (part of) many stored elements. The opposite is a *local* memory, in which different stored memory elements are stored in different memory locations, and each memory location holds (part of) only one stored element. The Hopfield memory is distributed, while the Hamming memory and the usual Random Access Memory (RAM) are local.

The *(worst-case) capacity* of a memory is the maximum number of any arbitrary permissible vectors that can be successfully stored in the memory. By permissible vectors we mean vectors obeying the possible restrictions which are posed in each specific situation. For example, in the usual RAM memory, the permissible vectors are binary vectors with length not exceeding the actual word length, and the worst-case capacity is the number of different memory locations. In some versions of the Hopfield memory, again, we may require the stored vectors to be almost orthogonal.

The most obvious associative memory has the stored vector pairs (u^i, v^i), for $i = 1$, 2, ..., m, stored in a RAM-type memory in which every vector has its own unique memory cell. (See Figure 2.) For retrieval, the input x is compared with each u^i, and the best matching vector u^j is selected. The corresponding vector v^j is given as output. This trivial associative memory can be realized using a neural network that picks the maximum in a set of values. Such a network is called a *winner-take-all* (WTA) network. The Hamming memory is essentially a neural realization of this trivial associative memory.

2.3 Introducing Zero Valued Elements

The associative memories are in most studies taken to receive as input and give as output only binary (0,1) or bipolar (± 1) vectors. There is a straightforward correspondence between the binary and the bipolar vectors, so we restrict our discussion to bipolar vectors.

One intent of the research described in this thesis is to demonstrate that the use of "don't know" elements is powerful both in the input and in the output, while the stored vectors are still bipolar. The obvious way to extend bipolar (± 1) vectors to allowing also "don't know" elements is to add the value zero to the possible element values, thus making the vectors ternary $(-1, 0, +1)$. This is handy especially for the input vectors, where the standard way of handling an element for which the value is unknown is to choose randomly -1 or $+1$ for it. This can naturally give very bad results if the random choices turn out unfavorably. The idea of using zero valued elements in the input has been proposed earlier [14, 86]. We allow zero valued elements also in the output, meaning that -1 or $+1$ could both be chosen, but the memory did not prefer either of them. This could be useful if several stored vectors are at minimum distance from the input vector. If the input vector to an autoassociative memory is $(1, 1, \ldots, 1)$, and both $(-1, 1, 1, \ldots, 1)$ and $(1, -1, 1, \ldots, 1)$ are the closest stored vectors, then a suitable output could rather be $(0, 0, 1, \ldots, 1)$. Note that this cannot be a stored vector, as it is not bipolar.

The versions with bipolar output vectors are in the following referred to as the *standard* models. The emphasis in this thesis is on standard memories with ternary input vectors. The stored vectors are, however, always assumed to be bipolar.

There are three obvious ways to handle situations with several possible output vectors: We can take as the result

1. one of these vectors,
2. a partial or total "don't know" answer (some elements are zero or the result is $\overline{0}$),
3. a linear combination of the possible vectors.

For Hopfield networks, the first two alternatives will be applied; for Hamming networks, all three alternatives will be explored.

3 Structure and Convergence of Hopfield Memories

The *Hopfield memory network* has attracted the attention of many physicists because of its analogy in behavior with so called Ising spin models in the theory of magnetism. It has got its name from John Hopfield, a physicist who made this model well-known and investigated some of its basic properties in 1982 [64]. Nevertheless, the model is an old one related to work by Amari, Anderson, Kohonen, Nakano, and Willshaw [2, 10, 11, 76, 91, 92, 128]. Also the Little model [83, 84] is related. Hopfield has also proposed a Hopfield network with real-valued nodes [65]. He has also suggested to use the Hopfield network for solving combinatorial optimization problems [66], but this approach is not without complications [17, 129, 131]. Kosko's Bidirectional Associative Memory [79] is a heteroassociative version of the Hopfield memory. Many more extensions of the memory model exist, see [62].

The fundamental properties and capabilities of Hopfield memories have been analyzed extensively in the literature (besides the references elsewhere in this chapter, see also, e.g., [1, 3, 4, 5, 7, 15, 16, 18, 21, 26, 41, 42, 43, 59, 68, 74, 117, 119, 122, 126]). The best books dealing with many aspects of Hopfield networks are by Kamp and Hasler [71] and by Hertz, Krogh and Palmer [62]. In contrast to these previous analyses we will also handle ternary vectors. We examine three themes: the advantage of ternary vectors over binary vectors, the convergence time, and the computational complexity of assessing the quality of a given memory network.

Section 3.1 describes the model and discusses the capacity of the model, Section 3.2 deals with the question of ternary vectors versus binary vectors, and Section 3.3 studies the convergence times. Chapter 4 then deals with the computational complexity of assessing the memory quality.

3.1 Architecture and Main Properties

The Hopfield memory is autoassociative. The memory network has n nodes, where n is the vector length, and all the nodes act as both input and output nodes. The set of possible node states is $\{-1, 0, 1\}$ or $\{-1, 1\}$, depending on whether zero is a possible vector element or not. Every node j is connected to every node i with an undirected edge with weight w_{ij}. The undirectionality means that the connection matrix $W = (w_{ij})$ is symmetric. The node function in the standard Hopfield memory is the *signum* function,

$$\mathrm{sgn}(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ -1, & \text{if } x < 0. \end{cases} \tag{1}$$

The value of sgn(0) is usually taken to be 1, sometimes it is taken to be -1 [82], and rather frequently it is taken to mean that the state of the node remains unchanged [12, 13, 22, 80, 99]. Note that there is no essential difference between choosing $\mathrm{sgn}(0) = -1$, or $\mathrm{sgn}(0) = 1$, since we can simulate the other choice by inputting $-x$ instead of x and treating output y as $-y$. We choose here $\mathrm{sgn}(0) = 1$.

When there are also zero valued elements in the output, we replace the signum function with a function also giving zero as output:

$$S_\eta(x) = \begin{cases} 1 & \text{if } x > \eta \\ 0 & \text{if } -\eta \leq x \leq \eta \\ -1 & \text{otherwise,} \end{cases} \tag{2}$$

where η is a parameter with some proper value for the current application. We call this parameter *tolerance*. We assume that the connection matrices have integer entries, so the argument of S_η is an integer, and therefore we assume that η is an integer. Formula (2) gives the usual signum function for $\eta = -1$. The value $\eta = 0$ corresponds to the signum function with the choice $\mathrm{sgn}(0) = 0$. The Hopfield network with node function S_η is here called the Hopfield memory with tolerance η. Hence, the standard Hopfield memory is a Hopfield memory with tolerance -1.

There are several different *storage rules*, i.e., methods for choosing the weights of the connections and the threshold values of the nodes. The best-known is the tensor product storage rule, also called the outer product rule, which we will present later in this section.

The updating can be either asynchronous or synchronous. We assume that a fair random order is employed in the asynchronous updating, so the results are nondeterministic, since we can end up in different output vectors depending on the order in which the nodes are updated. However, also a *sequential* update order is often studied in the literature: All nodes are updated one at a time in a specific order. In synchronous updating, the updating of the network state can be expressed as $x := S_\eta(Wx - \theta)$, where x is the network state and $\theta = (\theta_1, \theta_2, \ldots, \theta_n)$ is the vector of threshold values.

The computation is terminated when no state can any longer be changed: $S_\eta(Wy - \theta) = y$. We call such a y a *stable state* or a *stable vector*. Also the term fixed point is frequently encountered in the literature. The set of stable states is independent of whether the updating is synchronous or asynchronous. The stable vector reached upon convergence is the resulting output vector; thus it is bipolar in the standard Hopfield memory.

In asynchronous updating, the network converges always if the diagonal elements of the connection matrix are nonnegative [64]. In synchronous updating, we either reach convergence or enter a cycle of length two [50]. We can interpret such a result as a total "don't know" result or as a partial "don't know" result with the elements in which the two alternating states disagree put to zero. Convergence properties are studied in more detail in Section 3.3.

The Hopfield memory is not especially good. Firstly, we can get output vectors that are not stored vectors. These are called *spurious* stable vectors. A taxonomy of spurious stable vectors is given in [62, p. 24]. The simplest type of spurious stable vectors is the form $-u$ when u is a stored vector. Such vectors can be excluded by putting an additional element 1 in front of every vector. If we then get as output a vector beginning with 0, we interpret this as a "don't know" result, and if we get as output a vector of type $(-1, x_1, x_2, \ldots, x_n)$, we interpret this as $(-x_1, -x_2, \ldots, -x_n)$.

Secondly, the capacity is low: If too many vectors are stored, some of them cannot be retrieved from memory. In the general case, regardless of the storage rule, the worst-case capacity for the standard Hopfield memory when the connection matrix has at least one zero on the diagonal is 1 [19, 27, 89]. The Hamming memory discussed later makes more efficient use of nodes and connections. However, the Hopfield memory is inherently fault tolerant as the network is still giving reasonable results even if some nodes are out of operation.

Despite the weaknesses of the Hopfield memory, we investigate it here, because the ideas in the Hopfield memory provide the basis for many other models and have relevance also in other contexts, such as Ising spins in physics. Also, advanced storage rules and extensions of the model give higher capacity and better average-case performance.

Let u^1, u^2, …, u^m be the vectors we store. Recall that all stored vectors are bipolar. The tensor product, or outer product, storage rule chooses

$$W = \sum_{i=1}^{m} (u^i (u^i)^{\mathrm{T}} - I) \quad \text{and} \quad \theta = \overline{0}. \tag{3}$$

The rule expresses a Hebbian [60] 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.

Note that

$$(Wu^i)_r = (n-1)u_r^i + \sum_{j \neq i} \sum_{s \neq r} u_s^i u_s^j u_r^j. \tag{4}$$

Thus, if the vectors are random (different elements are -1 or $+1$ with equal probability independently of each other), the first term, the "signal", is $(n-1)u_r^i$, and the second term, the "noise", has 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 [64, 86]. For a more formal treatment, assume that the stored vectors are random in the way described above and denote by $P(m, n)$ the probability that all m stored vectors of length n are stable vectors in the network. Then the *probabilistic capacity* of a memory is $C(n)$, if for every $\lambda \in]0, 1[$ it holds that $\lim_{n\to\infty} P(m, n) = 1$ whenever $m \leq (1-\lambda)C(n)$, and $\lim_{n\to\infty} P(m, n) = 0$ whenever $m \geq (1+\lambda)C(n)$ [118, 120]. The probabilistic capacity of the network when using the tensor product storage rule is $n/(4 \ln n)$ [80, 86, 120]; see also [113, 123]. Accepting a fraction of erroneous elements in the result, we achieve a capacity of at most $0.138n$ [9].

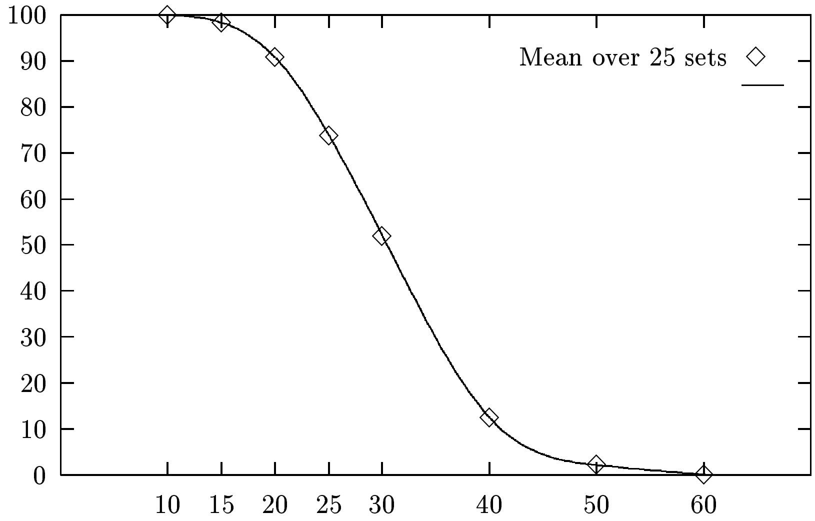

Figure 3: The points in the plot illustrate what percentage of the vectors stored with the tensor product storage rule were actually stable vectors in the network. The vector length (n) was 200. The number of vectors stored (m) was 10, 15, 20, 25, 30, 40, 50, and 60. For each m, the plotted value is the mean over 25 sets of random bipolar vectors.

As an illustration of the limited applicability of the tensor product storage rule, we stored random vectors of length 200 and checked how many of the stored vectors became stable. The result is shown in Figure 3. Note that the probabilistic capacity is $200/(4 \ln 200) \approx 9.4$; in the experiment, storing 10 vectors always resulted in all stored vectors being stable.

From formula (4) we see that if the stored vectors are almost orthogonal, then the "noise" term is very near to zero. This type of restriction has been studied, for instance, by Dembo [27]. In fact, for those n for which there are Hadamard matrices[2] of order n [56, Ch. 14] (for instance, $n = 2^k$ for some $k \geq 1$), up to $n-1$ orthogonal vectors can be stored. This is easy to prove: Let u^1, u^2, ..., u^n be n orthogonal vectors in $\{-1,1\}^n$.

[2] A Hadamard matrix H of order n is an $n \times n$ matrix with entries from $\{-1,1\}$ such that any two rows (or columns) of H are orthogonal.

Then $\sum_{i=1}^{n} u^i(u^i)^{\mathrm{T}} = nI$, as for every $x = \sum_{j=1}^{n} \alpha_j u^j$ we have

$$(\sum_{i=1}^{n} u^i(u^i)^{\mathrm{T}})x = \sum_{i=1}^{n} u^i((u^i)^{\mathrm{T}}x) = \sum_{i=1}^{n} u^i \left(\sum_{j=1}^{n} \alpha_j (u^i)^{\mathrm{T}} u^j\right) = n\sum_{i=1}^{n} \alpha_i u^i = nx = nIx.$$

Assume w.l.o.g. that vectors $u^1, u^2, \ldots, u^{n-1}$ are stored using the tensor product storage rule. Then

$$W = \sum_{i=1}^{n-1}(u^i(u^i)^{\mathrm{T}} - I) = \sum_{i=1}^{n} u^i(u^i)^{\mathrm{T}} - u^n(u^n)^{\mathrm{T}} - (n-1)I = I - u^n(u^n)^{\mathrm{T}}.$$

But this means that $Wu^i = u^i - u^n(u^n)^{\mathrm{T}}u^i = u^i$ for $i = 1, 2, \ldots, n-1$, and the claim is proved.

Baum et. al. [14] and Amari [6] 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.

Venkatesh and Psaltis [120] discuss so called spectral algorithms as an alternative to the tensor product storage rule. The idea is to construct a weight matrix with the stored vectors as eigenvectors, but possibly with different eigenvalues. The construction is more involved than the tensor product storage rule, but the probabilistic capacity is better — it is n. The maximum possible probabilistic capacity for Hopfield type networks with asymmetric connection matrices is $2n$ [23, 118, 121]; see also [44].

Our theoretical results in Section 3.3 and Chapter 4 are independent of the storage rule, but we will use the tensor product storage rule in our illustrative simulations, because it is an easy and commonly used method.

3.2 Role of Zero Valued Elements

The classical studies of Hopfield networks use binary vectors. We study the extension to ternary input and output vectors. Ternary input vectors give a natural way to use the Hopfield memory as a heteroassociative memory: The vectors we store are concatenations of the input and output vector pairs, and retrieval is achieved by giving as input a vector consisting of (almost) the input vector part with the rest filled up with zero valued elements.

We will distinguish between three cases for the type of input and output vectors:

1. There are no "don't know" elements (both input and output vectors are bipolar vectors). We refer to this case as *bipolar* (standard) Hopfield memories.

2. There can be "don't know" elements in the input (the input vectors are ternary vectors and the output vectors are bipolar vectors). We refer to this case as *ternary standard* Hopfield memories.

3. There can be "don't know" elements in both the input and the output (both input and output vectors are ternary vectors). We refer to this case as *ternary* Hopfield memories.

The fourth possibility with bipolar input and ternary output is not so interesting: In that case we can use a ternary Hopfield memory. In the rest of this section we will explore the possible advantages of ternary vectors over binary vectors.

First we demonstrate that ternary vectors are better than bipolar vectors in the input. Assume u is a stored vector and x is an input vector with k elements differing from the elements of u. We can assume w.l.o.g. that these are the k first elements. For bipolar vectors, this means that the distance between u and x is $2k$ in 1-metric (see Appendix I), and that

$$\operatorname{sgn}(\sum_{j=1}^{n} w_{ij}x_j - \theta_i) = \operatorname{sgn}(\sum_{j=1}^{n} w_{ij}u_j - \theta_i - 2\sum_{j=1}^{k} w_{ij}u_j). \tag{5}$$

The input to node i is disturbed by an amount $2|\sum_{j=1}^{k} w_{ij}u_j|$. In the case with ternary vectors and the k first elements of x being zero, the distance between u and x is k and

$$\operatorname{sgn}(\sum_{j=1}^{n} w_{ij}x_j - \theta_i) = \operatorname{sgn}(\sum_{j=1}^{n} w_{ij}u_j - \theta_i - \sum_{j=1}^{k} w_{ij}u_j), \tag{6}$$

so the disturbance is half of that of the bipolar case. Consequently, the correct result u is reached for ternary vectors when the correct result is reached for bipolar vectors, and may additionally be reached in cases where the result fails for bipolar vectors.

The situation is more complicated in the output: Allowing zero valued elements in the output has both advantages and disadvantages. The capacity decreases with increased tolerance. On the other hand, the retrieval can become better: If a vector x was not converging to u when tolerance η was -1 (or 0), then increasing η to

$$\eta_{\max} = \max\{\, |\sum_{j=1}^{n} w_{ij}(u_j - x_j)| \mid i = 1, \ldots, n\,\} \tag{7}$$

prevents $S_{\eta_{\max}}(x)$ from having any elements opposite to the elements of u; however, several zeros might be introduced.

What value of η we should choose depends on our measure of the quality of recall. To put it drastically: A trivial solution denying any element to get the opposite value of what it should be is to put η arbitrarily large — then the output is always $\overline{0}$. Intuitively, we usually prefer a "don't know" element to a wrong element, but trying to avoid wrong elements must not introduce too many "don't know" elements. The distance between the output vector and the desired stored vector is a measure of the quality of the recall; different metrics, however, give different results. The p-metric gives weight 2^p to wrong elements and weight 1 to "don't know" elements. If $p = 1$, "don't know" elements are not especially much appreciated, if p is very large, we get the drastic situation described above with a preference for very large values of η.

The effect of "don't know" elements were tested with some simulations. Ten random bipolar vectors of length 200 were generated and stored in a Hopfield memory with the tensor product storage rule. All the vectors were stable for η up to 83. The input vector sets were generated by taking 250 replicates of each of the ten stored vectors and randomly changing 80 elements to either their opposite values or to zero. The input vector sets

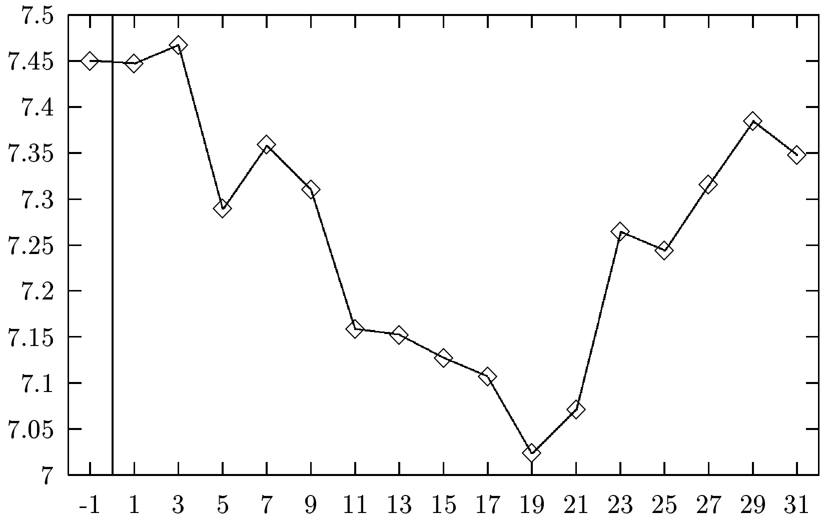

Figure 4: The mean Euclidean distances between the output vectors and the corresponding stored vectors from which the input vectors were generated for the standard Hopfield memory ($\eta = -1$) and for the ternary Hopfield memory with $\eta = 0, 1, \ldots, 31$. The update order was sequential, so the results for $\eta = 2k$ are equal to the results for $\eta = 2k + 1$ for all $k \geq 0$. Therefore, we display the results for odd η only.

were run through both a standard Hopfield memory and a ternary Hopfield memory with different tolerances values. The update order was sequential.

When using the input vector set with elements changed to zero, exactly the correct results were obtained for all tested tolerances ($\eta = 0, 1, \ldots, 31$). The results for the input vector set with elements changed to their opposite values are shown in Figure 4. As can be seen from the figure, there are not especially big differences between the results for different tolerance values. Also, the usefulness of zero valued elements in the output is diminished by the difficulty to establish the best tolerance value.

The discussion and the experiments in this section show that it is important to have "don't know" elements available in the input, but that having "don't know" elements in the output does not provide much advantage. For the sake of completeness, we still express our results in Section 3.3 and Chapter 4 for both bipolar and ternary vector cases.

3.3 Convergence

The general convergence behavior of Hopfield networks is well studied, see e.g. [71]. The contributions in this section are improvements on the worst-case bounds on the convergence time and extensions of the results from binary to ternary vectors.

The following theorems ensure that the network converges under asynchronous updating when the diagonal elements of the connection matrix are nonnegative. The theorems give also bounds on the convergence time. The expression *integer parameters* means here that the connection matrix and the threshold vector have only integer elements. Recall that the connection matrix is symmetric.

First we look at the standard model. Remembering that the output of a node is bipolar, we know that in spite of zero elements in the input, each node has state $+1$ or -1 after an update. Therefore, we state the theorem assuming bipolar input vectors. The bound is an improvement on results obtained by Fogelman, Goles et al. [40, 48, 50, 52]. Their corresponding bound for vectors in $\{0,1\}^n$ is $\sum_{i,j} |w_{ij}| + 2\sum_i |\theta_i|$. The same bound is valid also for vectors in $\{-1,1\}^n$. The trick in their proof is that the threshold values are changed to get rid of situations where $\sum_{j=1}^n w_{ij}x_j - \theta_i = 0$. Our approach is different: We modify the corresponding Lyapunov function. A Lyapunov function is a monotonic function associated with the system under consideration; Hopfield [64] calls it "energy." Lyapunov functions for neural networks are studied, for instance, in [54, 75].

Theorem 1 *Given any bipolar input vector, the asynchronous standard Hopfield memory with integer parameters and nonnegative diagonal elements in the connection matrix converges to a stable vector in at most*

$$\frac{M + \sum_i |\theta_i - e_i|}{1 + \min_k w_{kk}}$$

changing update steps, i.e., update steps in which the state of a node is really changed. Here M is the sum of the $\lfloor n^2/4 \rfloor$ largest elements in the multiset $\{\, |w_{ij}| \mid i < j \,\}$ and e_i is 1, if $\sum_j w_{ij} - \theta_i$ is even, and 0 otherwise.

Proof Choose the Lyapunov function (in the following called energy function)

$$E(x(t)) = -\sum_{i,j} w_{ij} x_i(t) x_j(t) + 2\sum_i (\theta_i - e_i) x_i(t).$$

Let us study a changing update step and for simplicity denote $x' = x(t+1)$ and $x = x(t)$. As the updating is asynchronous, there exists a k such that $x'_k \neq x_k$, and $x'_j = x_j$ for all $j \neq k$. Thus the difference in energy is

$$\begin{aligned}
\Delta E &= E(x') - E(x) \\
&= \sum_{i,j} w_{ij} x_i x_j - \sum_{i,j} w_{ij} x'_i x'_j - 2\sum_i (\theta_i - e_i)(x_i - x'_i) \\
&= 2\sum_{j \neq k} w_{kj}(x_k x_j - x'_k x'_j) + w_{kk}((x_k)^2 - (x'_k)^2) - 2(\theta_k - e_k)(x_k - x'_k) \\
&= 2\sum_j w_{kj} x_j (x_k - x'_k) - 2w_{kk} x_k (x_k - x'_k) + w_{kk}(x_k + x'_k)(x_k - x'_k) \\
&\quad - 2(\theta_k - e_k)(x_k - x'_k) \\
&= \Big[2(\sum_j w_{kj} x_j - \theta_k + e_k) - w_{kk}(x_k - x'_k)\Big](x_k - x'_k).
\end{aligned}$$

The purpose of the e_k-values is to ensure that the expression $\sigma_k = \sum_j w_{kj} x_j - \theta_k + e_k$ is never zero; if it could be zero, there could be changing update steps that do not decrease the energy. The following two cases occur:

1. If $x_k = -1$ and $x'_k = \text{sgn}(\sigma_k - e_k) = 1$, we have $\sigma_k \geq 1$; thus $\Delta E \leq -4 - 4w_{kk}$.

2. If $x_k = 1$ and $x'_k = \text{sgn}(\sigma_k - e_k) = -1$, we have $\sigma_k \leq -1$; thus $\Delta E \leq -4 - 4w_{kk}$.

Consequently, whenever the state of node k changes, the energy decreases at least by an amount of $4 + 4w_{kk}$.

Now we give an upper bound on the possible energy difference between the first and the last vector of any sequence of update steps. We consider the two terms of $E(x)$ separately: $E_1(x) = -\sum_{i,j} w_{ij} x_i x_j$ and $E_2(x) = 2\sum_i (\theta_i - e_i) x_i$. But

$$E_1(x) = -2\sum_{i<j} w_{ij} x_i x_j - \sum_i w_{ii} x_i^2 = -2(\sum_{S_+(x)} w_{ij} - \sum_{S_-(x)} w_{ij}) - \sum_i w_{ii},$$

where $S_+(x) = \{\, (i,j) \mid i < j \text{ and } x_i x_j = 1 \,\}$ and $S_-(x) = \{\, (i,j) \mid i < j \text{ and } x_i x_j = -1 \,\}$. Now the minimum value of $E_1(x)$ is achieved for some x; call this vector $x_{\min}$. Changing

k elements of $x_{\min}$ changes those $x_i x_j$-values for which one of the factors change and the other not. Thus $k(n-k)$ values move from S_+ to S_- or from S_- to S_+. The expression $k(n-k)$ achieves its maximum value for $k = n/2$, and then the maximum value is $n^2/4$. Thus the maximum change in the value of E_1 is four times the sum of the $\lfloor n^2/4 \rfloor$ largest $|w_{ij}|$-values ($i < j$). (It is *four* times, because there is a factor 2 in the expression $E_1(x)$, and the values w_{ij} are turned to their *opposite* values.) For the second term $E_2(x)$, the minimum value is $-2\sum_i |\theta_i - e_i|$ and the maximum value is $2\sum_i |\theta_i - e_i|$. Thus the energy difference is bounded by $4M + 4\sum_i |\theta_i - e_i|$, where M is the sum of the $\lfloor n^2/4 \rfloor$ largest elements in the multiset $\{\, |w_{ij}| \mid i < j \,\}$.

The bound on the number of changing update steps follows. As the asynchronous updating is fair, the computation eventually converges. □

Note that $M \leq \frac{1}{2}\sum_i \sum_{j \neq i} |w_{ij}|$, so our bound is about half the previously known bound. With the tensor product storage rule, we get $|w_{ij}| \leq m$, and hence the upper bound on the number of changing update steps is $mn^2/4 + n$, where m is the number of stored vectors and n is their length.

The bound in the theorem can be reached, for instance, by a matrix W and threshold vector θ with all elements zero, since then $\overline{1}$ is the only stable vector and beginning from vector $-\overline{1}$ the network will need exactly the number of changing update steps given by this bound. This is, however, an unnatural example. The bound is a worst-case *upper* bound, and simulation results show that the number of changing update steps is in practice at most slightly greater than the number of elements differing between the input and the output vectors (see Table 1).

If some w_{ij}- and θ_i-values are exponential in n, then the convergence time can actually be exponential in n [53]; see also [51, 55]. If W has negative elements on the diagonal, the network is not assured to converge. In sequential updating, negative diagonal elements can cause the network to enter a cycle (which may be long), see [71, p. 26].

We study now the situation with ternary output vectors. The next theorem shows that the network always converges to a stable vector also when node function S_η is used. The

Table 1: Number of changing update steps and total number of update steps when 20 random bipolar vectors of length 200 were stored according to the tensor product storage rule (all were stable, $M = 56984$, $e_i = 1$ for all i) and 1000 random bipolar input vectors were fed through the asynchronous standard Hopfield memory. The bound is the upper bound provided by Theorem 1.

# u.steps	changing	total
mean	146	2652
maximum	219	8952
bound	57184	

convergence for node functions of this type has been proved already in [30], but without any bounds on the convergence time. Our bound on the worst-case convergence time is about four times the bound for the case with bipolar vectors.

Theorem 2 *Given any ternary input vector, the asynchronous ternary Hopfield memory with integer parameters, nonnegative diagonal elements in the connection matrix, and tolerance $\eta \geq 0$ converges to a stable vector in at most*

$$\frac{4M + \sum_i w_{ii} + 4\sum_i |\theta_i| + n(2\eta+1)}{1 + \min_k w_{kk}}$$

changing update steps, where M is the sum of the $\lfloor n^2/4 \rfloor$ largest elements in the multiset $\{\, |w_{ij}| \mid i < j \,\}$.

Proof The proof is analogous to the proof of Theorem 1. The energy function is now

$$E(x(t)) = -\sum_{i,j} w_{ij}x_i(t)x_j(t) + 2\sum_i \theta_i x_i(t) + \rho \sum_i |x_i(t)|,$$

where the value of ρ will be determined later in the proof.

Let us study a changing update step and for simplicity denote $x' = x(t+1)$ and $x = x(t)$. As the updating is asynchronous, there exists a k such that $x'_k \neq x_k$, and

$x'_j = x_j$ for all $j \neq k$. Thus the difference in energy is

$$\begin{aligned}
\Delta E &= E(x') - E(x) \\
&= \sum_{i,j} w_{ij} x_i x_j - \sum_{i,j} w_{ij} x'_i x'_j + 2\sum_i \theta_i (x'_i - x_i) + \rho \sum_i (|x'_i| - |x_i|) \\
&= \Big[2(\sum_j w_{kj} x_j - \theta_k) - w_{kk}(x_k - x'_k) - \rho(x_k + x'_k)\Big](x_k - x'_k).
\end{aligned}$$

For brevity, let us again denote $\sigma_k = \sum_j w_{kj} x_j - \theta_k$. Note that $x'_k = S_\eta(\sigma_k)$. We now have six cases:

1. If $x_k = -1$ and $x'_k = 0$, we have $|\sigma_k| \leq \eta$; thus $\Delta E \leq 2\eta - w_{kk} - \rho$.
2. If $x_k = -1$ and $x'_k = 1$, we have $\sigma_k \geq \eta + 1$; thus $\Delta E \leq -4\eta - 4 - 4w_{kk}$.
3. If $x_k = 0$ and $x'_k = -1$, we have $\sigma_k \leq -(\eta + 1)$; thus $\Delta E \leq -2\eta - 2 - w_{kk} + \rho$.
4. If $x_k = 0$ and $x'_k = 1$, we have $\sigma_k \geq \eta + 1$; thus $\Delta E \leq -2\eta - 2 - w_{kk} + \rho$.
5. If $x_k = 1$ and $x'_k = -1$, we have $\sigma_k \leq -(\eta + 1)$; thus $\Delta E \leq -4\eta - 4 - 4w_{kk}$.
6. If $x_k = 1$ and $x'_k = 0$, we have $|\sigma_k| \leq \eta$; thus $\Delta E \leq 2\eta - w_{kk} - \rho$.

Clearly, we have to choose ρ in such a way that $2\eta - w_{kk} - \rho < 0$, $-4\eta - 4 - 4w_{kk} < 0$, and $-2\eta - 2 - w_{kk} + \rho < 0$. To get the best possible bound on the number of update steps, the values should be as small as possible (large in absolute value). This is obtained by choosing $\rho = 2\eta + 1$, and the inequalities become $-1 - w_{kk} < 0$, $-4\eta - 4 - 4w_{kk} < 0$, and $-1 - w_{kk} < 0$. The inequalities are obviously satisfied when $\eta > -1 - w_{kk}$. The minimum decrease in energy in one changing update step is thus $1 + \min_k w_{kk}$.

For an upper bound on the possible energy difference, we proceed as in Theorem 1. Consider each term of the energy function separately. Now

$$E_1(x) = -\sum_{i,j} w_{ij} x_i x_j = E'_1(x) - \sum_{T(x)} w_{ii} = -2(\sum_{S_+(x)} w_{ij} - \sum_{S_-(x)} w_{ij}) - \sum_{T(x)} w_{ii},$$

where $S_+(x)$ and $S_-(x)$ are defined as in Theorem 1 and $T(x) = \{\, i \mid x_i \neq 0 \,\}$. The minimum value of E'_1 is achieved for some x; call this vector $x_{\min}$. Now $x_{\min}$ can include zero valued elements. Denote by d the number of nonzero elements in $x_{\min}$; i.e., $d = |T(x_{\min})|$. Trying now to maximize E_1', we make two observations. Firstly, there is no use to change a -1 or $+1$ to zero, because a greater difference would be obtained by making a change to the opposite value. Secondly, a zero cannot be changed to a -1 or $+1$, because if some higher value for E_1' were obtained by changing a zero to, say, $+1$, then the energy value E_1' would decrease by changing it to -1, which is a contradiction to $x_{\min}$ giving the minimum E_1'-value. Thus the only changes we have to consider are those with k elements changed to their opposite values. This moves $k(d-k)$ nodes from S_+ to S_- or from S_- to S_+. But $d \leq n$, so analogously as in Theorem 1, we get the maximum change in the value E_1' bounded by four times the sum of the $\lfloor n^2/4 \rfloor$ largest $|w_{ij}|$-values ($i < j$). The maximum difference in $\sum_{T(x)} w_{ii}$ is $\sum_i w_{ii}$, in the second energy term is $4\sum_i \theta_i$, and in the third term is $n\rho = n(2\eta + 1)$.

The bound on the number of changing update steps follows. As the asynchronous updating is fair, the computation eventually converges. □

With the tensor product storage rule, the upper bound on the number of changing update steps is $mn^2 + n(2\eta + 1)$, where m is the number of stored vectors and n is the vector length.

Taking $\eta = 0$, the bound in the theorem can again be reached by a matrix W with all elements zero, as for instance $\overline{1}$ needs n update steps to reach the stable resulting vector $\overline{0}$. As for Theorem 1, the bound is a worst-case upper bound, and simulation results show only slightly longer convergence times than for the standard Hopfield memory (see Table 2).

The corresponding results can be obtained also for synchronous updating.

Table 2: Number of changing update steps and total number of update steps when the same vectors as in Table 1 ($m = 20$, $n = 200$) were stored according to the tensor product storage rule ($\sum_{i,j} |w_{ij}| = 139680$) and 1000 random ternary input vectors (the probability for -1 was 0.4, the probability for zero was 0.2, and the probability for $+1$ was 0.4) were fed through the asynchronous Hopfield memory with tolerance $\eta = 0$ and $\eta = 10$. The bound is the upper bound provided by Theorem 2.

	$\eta = 0$		$\eta = 10$	
# u.steps	changing	total	changing	total
mean	173	2845	217	3386
maximum	268	9018	351	10618
bound	228136		232136	

Theorem 3 *Given any bipolar input vector, the synchronous standard Hopfield memory with integer parameters reaches either a stable vector or a cycle of length two in at most*

$$\frac{1}{2}\Big(\sum_{i,j} |w_{ij}| + 3\sum_{i} |\theta_i - e_i| - n\Big)$$

update steps, where e_i is 1, if $\sum_j w_{ij} - \theta_i$ is even, and 0 otherwise.

Proof Choose energy function

$$E(x(t), x(t-1)) = -\sum_{i,j} w_{ij} x_i(t) x_j(t-1) + \sum_{i} (\theta_i - e_i)\,(x_i(t) + x_i(t-1))\,.$$

Thus the difference in energy is

$$\begin{aligned} \Delta_t E &= E(x(t+1), x(t)) - E(x(t), x(t-1)) \\ &= -\sum_{i} \Big[(\sum_{j} w_{ij} x_j(t) - \theta_i + e_i)\,(x_i(t+1) - x_i(t-1)) \Big]. \end{aligned}$$

As in Theorem 1, the e_i-values ensure that the expression $\sum_j w_{ij} x_j(t) - \theta_i + e_i$ is never zero.

If $x(t+1) = x(t-1)$, we have reached a stable vector or a cycle of length two. On the other hand, if $x(t+1) \neq x(t-1)$, there is some k for which $x_k(t+1) \neq x_k(t-1)$. Clearly, $\sum_j w_{kj}x_j(t) - \theta_k + e_k$ has the same sign as $x_k(t+1)$; thus $x_k(t+1) \neq x_k(t-1)$ implies $\Delta_t E \leq -2$.

We see that

$$1 \leq x_i(t) \left(\sum_j w_{ij}x_j(t-1) - \theta_i + e_i \right) \leq \sum_j |w_{ij}| + |\theta_i - e_i|.$$

Summing over i we get

$$n \leq \sum_{i,j} w_{ij}x_i(t)x_j(t-1) - \sum_i (\theta_i - e_i)x_i(t) \leq \sum_{i,j} |w_{ij}| + \sum_i |\theta_i - e_i|,$$

thus giving

$$-\sum_{i,j} |w_{ij}| - 2\sum_i |\theta_i - e_i| \leq E(x(t), x(t-1)) \leq -n + \sum_i |\theta_i - e_i|.$$

From this we get $\sum_{i,j} |w_{ij}| + 3\sum_i |\theta_i - e_i| - n$ as an upper bound on the possible energy difference. The bound on the update steps follows. □

Note that the update steps counted in the theorem do not include the one or two possible additional steps that have to be performed in order to check whether a stable state or a cycle of length two is reached.

The result in Theorem 3 has been obtained by Goles [49] when restricted to $\theta = \overline{0}$. He has also shown that networks with connection matrices of type

$$W = \begin{pmatrix} 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 1 \end{pmatrix}$$

(and $\theta = \overline{0}$) achieve this bound on the convergence time when the input vector is $(1, -\overline{1})$. Generally, this bound is again a worst-case upper bound, and simulation results show very

Table 3: Number of update steps when the same vectors as in Table 1 ($m = 20$, $n = 200$) were stored according to the tensor product storage rule and the same 1000 random bipolar input vectors as in Table 1 were fed through the synchronous standard Hopfield memory. The number of steps needed to check whether a stable state or a cycle of length two is reached is excluded. Of 1000 input vectors, 206 were such that a cycle of length two was the final result. The bound is the upper bound provided by Theorem 3.

mean	13.7
maximum	44
bound	70040

fast convergence (see Table 3). Komlós and Paturi [78] have shown that for less than $n/(4 \ln n)$ random stored vectors of sufficient length, an input vector sufficiently close to a stored vector will with high probability converge to that stored vector in $O(\log \log n)$ synchronous steps. If the $n \times n$-matrix W is positive definite on the set $\{-1, 0, 1\}^n$, the network converges always to a stable vector [49, 50] (see also [47]). Convergence times for synchronous updating are additionally studied in [48, 51].

Theorem 4 *Given any ternary input vector, the synchronous ternary Hopfield memory with integer parameters and tolerance $\eta \geq 0$ reaches either a stable vector or a cycle of length two in at most*

$$2 \sum_{i,j} |w_{ij}| + 6 \sum_{i} |\theta_i| + n(2\eta + 1)$$

update steps.

Proof Choose energy function

$$\begin{aligned} E(x(t), x(t-1)) &= -\sum_{i,j} w_{ij} x_i(t) x_j(t-1) + \sum_i \theta_i (x_i(t) + x_i(t-1)) \\ &\quad + (\eta + 1/2)(|x_i(t)| + |x_i(t-1)|). \end{aligned}$$

Thus the difference in energy is

$$\Delta_t E = E(x(t+1), x(t)) - E(x(t), x(t-1)) =$$
$$-\sum_i \Big[(\sum_j w_{ij} x_j(t) - \theta_i)(x_i(t+1) - x_i(t-1)) + (\eta + 1/2)(|x_i(t+1)| - |x_i(t-1)|) \Big].$$

If $x(t+1) = x(t-1)$, we have reached a stable vector or a cycle of length two. On the other hand, if $x(t+1) \neq x(t-1)$, there is some k for which $x_k(t+1) \neq x_k(t-1)$. For brevity, let us denote $\sigma_k(t) = \sum_j w_{kj} x_j(t) - \theta_k$. We now have the following possibilities:

1. If $x_k(t+1) = 1$, then $x_k(t-1) \leq 0$ and $\sigma_k(t) \geq \eta + 1$;
   thus $\Delta_t E \leq -(\eta+1)(1 - x_k(t-1)) + (\eta + 1/2)(1 + x_k(t-1)) \leq -1/2$.

2. If $x_k(t+1) = 0$, then $x_k(t-1) \neq 0$ and $|\sigma_k(t)| \leq \eta$;
   thus $\Delta_t E \leq \eta - (\eta + 1/2) = -1/2$.

3. If $x_k(t+1) = -1$, then $x_k(t-1) \geq 0$ and $\sigma_k(t) \leq -\eta - 1$;
   thus $\Delta_t E \leq -(-\eta - 1)(-1 - x_k(t-1)) + (\eta + 1/2)(1 - x_k(t-1)) \leq -1/2$.

We see that

$$0 \leq x_i(t) \left(\sum_j w_{ij} x_j(t-1) - \theta_i \right) - |x_i(t)|(\eta + 1/2) \leq \sum_j |w_{ij}| + |\theta_i|.$$

Summing over i we get

$$0 \leq \sum_{i,j} w_{ij} x_i(t) x_j(t-1) - \sum_i \theta_i x_i(t) - \sum_i |x_i(t)|(\eta + 1/2) \leq \sum_{i,j} |w_{ij}| + \sum_i |\theta_i|,$$

thus giving

$$-\sum_{i,j} |w_{ij}| - 2 \sum_i |\theta_i| \leq E(x(t), x(t-1)) \leq \sum_i |\theta_i| + n(\eta + 1/2).$$

From this we get $\sum_{i,j} |w_{ij}| + 3 \sum_i |\theta_i| + n(\eta + 1/2)$ as an upper bound on the possible energy difference. The bound on the update steps follows. □

Simulation results show again very fast convergence (see Table 4).

Note that, contrary to asynchronous updating, the results for synchronous updating do not require nonnegative diagonal elements in the connection matrix.

The convergence behavior of the Hopfield network is summarized in Table 5.

Table 4: Number of update steps when the same vectors as in Table 1 ($m = 20$, $n = 200$) were stored according to the tensor product storage rule and the same 1000 random ternary input vectors as in Table 2 were fed through the synchronous Hopfield memory with tolerance $\eta = 0$ and $\eta = 10$. The number of steps needed to check whether a stable state or a cycle of length two is reached is excluded. Of 1000 input vectors, 184 and 143, respectively, were such that a cycle of length two was the final result. The bound is the upper bound provided by Theorem 4.

	$\eta = 0$	$\eta = 10$
mean	14.7	18.0
maximum	46	59
bound	279560	283560

Table 5: The convergence behavior of Hopfield networks with different updating modes and different types of connection matrix diagonals. The connection matrix is symmetric. The expression "cycle of length ≤ 2" means cycle of length two or stable vector (i.e., cycle of length one).

	asyncronous updating	synchronous updating
any diagonal	cycle is possible	cycle of length ≤ 2
nonnegative diagonal	stable vector	cycle of length ≤ 2

4 Computational Complexity of Analyzing Hopfield Memories

Assume that we have stored the vectors u^1, u^2, ..., $u^m \in \{-1, 1\}^n$ in a Hopfield memory. This means that we have determined a connection matrix W and a threshold vector θ somehow, not necessarily with the tensor product storage rule, so that all the vectors u^1, u^2, ..., u^m are stable. In this chapter we examine how difficult it is to assess the quality of the resulting network by the means of a conventional computer (classical complexity classes). For an introduction to computational complexity and the notion of NP-completeness, see the book by Garey and Johnson [45].

Analyzing even the very basic properties of networks with asymmetric connection matrices is hard. For instance, Porat [100] has shown that the problem of determining whether every computation in an asymmetric network ends up in a stable vector is NP-hard, and Lipscomb et al. [46] have shown that even the problem of determining whether such a network has *any* stable vectors is NP-hard. Therefore, we restrict our study to symmetric connection matrices.

4.1 Quality Criteria for Hopfield Memories

We now turn to evaluating the quality of a network as an associative memory. The network is working well, if vectors near to a stored vector also converge to this stored vector. The set of all vectors guaranteed to converge to a stable vector u is the *attraction domain* of u. Especially, as attraction in asynchronous updating depends, in general, on the order in which the nodes are updated, this means, in fair random asynchronous updating, that we take as the attraction domain the set of those vectors that converge to the given stable vector regardless of the order in which the nodes are updated. The *attraction radius* of u (with respect to some given metric d) is the maximum r for which all vectors x such that $d(x, u) \leq r$ belong to the attraction domain. In other words, the radius is the minimum distance from u to the border of the attraction domain. For synchronous updating, we additionally define the *k-step attraction domain* of u as the set of vectors converging to u within k update steps. Correspondingly, the *k-step attraction radius* is the minimum distance from u to the border of the k-step attraction domain. The one-step attraction domain is called the *direct attraction domain*; the one-step attraction radius is called the *direct attraction radius.*

The quality of a network can be evaluated by examining the sizes and shapes of the attraction domains of the stored vectors. The natural property to examine is the attraction radius.

In asynchronous updating, an input vector can converge to different stable vectors depending on the update order. For asynchronous updating we can still get a sense of the quality by counting the number of spurious stable vectors: If there are many spurious stable vectors, we can expect the attraction radii of the stored vectors to be small. In Section 4.2 we show that it is hard to determine the number of spurious stable vectors.

Synchronous updating is deterministic, but can result in a cycle of length two. The stable vectors are the same for synchronous updating as for asynchronous updating. Note that for synchronous updating, the attraction domains can even consist of separate regions far from each other. For synchronous updating, we obtain results about the size of the attraction domain and about the attraction radius. In Section 4.3 we show that determining the size of the direct attraction domain is hard, and in Section 4.4 we see that it is easy to compute the direct attraction radius, but hard to compute the general attraction radius.

Limiting our study to symmetric connection matrices, more spurious stable vectors emerge with positive diagonal elements than with diagonal elements zero. As we will see, most of the problems we examine are hard for networks with zero diagonal, and thus they are hard also for the more general case with nonnegative diagonal elements. With this in mind, we always try to state the results using networks with connection matrices with zero diagonals. Consequently, we concentrate on the restricted class of networks with symmetric integer valued connection matrices with zero diagonal and threshold vector $\overline{0}$. This is the class of networks considered by Hopfield in [64]. We will use the name *basic* Hopfield memory for such networks.

These results show that the Hopfield memory not only has low capacity, but it is also hard to assess the quality of a given network. The last section, Section 4.5, provides a short summary of the results. The results in this chapter are to a large extent based on joint work with Pekka Orponen [37, 38, 39].

The applicability of our results to the tensor product storage rule is interesting. The following lemma implies that the results obtained (except Theorem 5 and Corollary 1) are valid also for matrices produced by the tensor product storage rule, when the same vector is allowed to be stored several times.

Lemma 1 *Given any symmetric integer valued connection matrix with zero diagonal, some multiple of it can be obtained using the tensor product storage rule.*

Proof Take the collection of all ± 1-vectors with $x_i = x_j = 1$. There are 2^{n-2} such vectors and the tensor product storage rule gives a matrix with all entries zero besides $w_{ij} = w_{ji} = 2^{n-2}$. Analogously, the collection of all ± 1-vectors with $x_i = -x_j = 1$ gives a matrix with all entries zero besides $w_{ij} = w_{ji} = -2^{n-2}$. Thus, repeating this procedure the required number of times for all entries of the desired matrix, we get the matrix $2^{n-2}W$. $\square$

It is impossible to obtain (a multiple of) any arbitrary matrix form, even with $|w_{ij}| \leq 1$, if the vectors are allowed to be stored only once. It is an open question whether our *NP*-completeness results are valid also restricted to connection matrices obtainable with the tensor product storage rule with each vector stored only once or polynomially many times.

In the following sections, we will frequently use vector notation for a sequence of integers $a_1, a_2, \ldots, a_k$; i.e., $a = (a_1, a_2, \ldots, a_k)$. From this we get the notation $a^{\mathrm{T}}x = \sum_{i=1}^{k} a_i x_i$ and $\overline{1}^{\mathrm{T}}x = \sum_{i=1}^{k} x_i$.

Now as we are ready to enter our collection of *NP*-completeness results, it might be helpful to recall the epithets we use for Hopfield networks:

- *standard* = bipolar output,
- *bipolar* or *bipolar standard* = bipolar input and output,
- *ternary standard* = ternary input and bipolar output,
- *ternary* = ternary input and output,
- *basic* = symmetric integer valued connection matrix with zero diagonal; threshold vector $\overline{0}$.

Consequently, standard means the union of bipolar and ternary standard Hopfield networks. Note that bipolar output implies node function sgn and ternary output implies node function S_η.

4.2 Number of Spurious Stable Vectors

Our first quality measure is the number of spurious stable vectors: the less spurious stable vectors, the better. Given a network with connection matrix W and threshold vector θ, it is easy to check that the stored vectors u^i, for $i = 1, 2, \ldots, m$, are all stable: Just compute $\operatorname{sgn}(Wu^i - \theta)$ for each one. Hence, computing the number of spurious stable vectors in a network is computationally equivalent to computing how many stable vectors it has in all. Unfortunately, this is hard.

We prove first a result that shows how hard the convergence properties are to determine when negative diagonal elements are allowed. For synchronous updating, the theorem means that it is hard to determine whether or not all input vectors end up in cycles of length two.

Theorem 5 *The problem "Given a standard Hopfield network with symmetric integer valued connection matrix and threshold vector $\overline{0}$; are there any stable vectors in the network?" is NP-complete.*

Proof Clearly, the problem is in *NP*. We prove that it is *NP*-hard by a reduction from Partition [45, p. 223].

Consider an instance of Partition: "Given positive integers $a_1, a_2, \ldots, a_k$; does there exist a vector $\xi \in \{-1, 1\}^k$ such that $a^{\mathrm{T}}\xi = 0$?" To reduce this to an instance of the problem under consideration, denote $c = \sum_{i=1}^k a_i$, and choose

$$W = \begin{pmatrix} -c & c & a_1 & a_2 & \cdots & a_k \\ c & -c & -a_1 & -a_2 & \cdots & -a_k \\ a_1 & -a_1 & 1 & 0 & \cdots & 0 \\ a_2 & -a_2 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_k & -a_k & 0 & 0 & \cdots & 1 \end{pmatrix}.$$

Clearly W can be constructed from $a_1, a_2, \ldots, a_k$ in polynomial time.

We now prove that there is a solution to the given Partition instance if and only if there is a stable vector in the network.

(if) If there is a stable vector of the form $(1, 1, \xi)$, where $\xi = (\xi_1, \ldots, \xi_k) \in \{-1, 1\}^k$, then by row 1 of matrix W, $a^{\mathrm{T}}\xi \geq 0$, and by row 2 of W, $-a^{\mathrm{T}}\xi \geq 0$. Hence $a^{\mathrm{T}}\xi = 0$.

Let us prove that the only possible stable vectors are of this form:

1. Form $(-1, -1, \xi)$ is impossible, because then rows 1 and 2 of matrix W would imply that $0 > a^{\mathrm{T}}\xi > 0$.

2. Form $(1, -1, \xi)$ is impossible, because then row 1 of matrix W would imply that $-2c + a^{\mathrm{T}}\xi \geq 0$. However, $a^{\mathrm{T}}\xi \leq c$, so $-2c + a^{\mathrm{T}}\xi \leq -c < 0$.

3. Form $(-1, 1, \xi)$ is impossible, because then row 1 of matrix W would imply that $2c + a^{\mathrm{T}}\xi < 0$. However, $a^{\mathrm{T}}\xi \geq -c$, so $2c + a^{\mathrm{T}}\xi \geq c > 0$.

(only if) If ξ is a solution to the PARTITION instance, then $x = (1, 1, \xi)$ is a stable vector. This concludes the proof. □

We can actually make a slightly stronger statement than in Theorem 5, but for this we need some additional notation. An integer valued function f belongs to the class $\#P$ 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 *#P-complete*, if it is in $\#P$, and any other function in $\#P$ can be computed by some deterministic polynomial time Turing machine that is allowed to access values of f at unit cost. To each *NP* decision problem there corresponds in a natural way a $\#P$ counting problem (the problem of counting accepting computation paths, i.e., solutions to the problem instance), and vice versa. We shall say, somewhat inaccurately, that an *NP* decision problem A is in $\#P$ (respectively $\#P$-complete) if the associated counting problem is in $\#P$ (respectively $\#P$-complete). A polynomial time transformation of one *NP* problem to another is *parsimonious* if it preserves the number of accepting computation paths for each instance. If the underlying *NP* problem A of a $\#P$-complete counting problem can be parsimoniously reduced to another *NP* problem B, then also the $\#P$ counting problem associated with B is $\#P$-complete. For a more extended discussion of these notions, see [45, pp. 168–169].

Corollary 1 *The problem "Given a standard Hopfield network with symmetric integer valued connection matrix and a threshold vector $\overline{0}$; how many stable vectors are there in the network?" is #P-complete.*

Proof The problem is clearly in $\#P$. Since PARTITION is $\#P$-complete [109], and the reduction in the proof of Theorem 5 is parsimonious, the claim follows. $\square$

We already know by Theorem 1 that the two negative diagonal elements in the proof of Theorem 5 are crucial for the possible existence of stable vectors. Nevertheless, even in the case of nonnegative diagonal elements it is hard to determine the *number* of stable vectors, as the following result shows.

Theorem 6 *The problem "Given a basic standard Hopfield network; does the network have at least three stable vectors?" is NP-complete.*

Proof Clearly, the problem is in NP. We prove that it is NP-hard by a reduction from a restricted version of PARTITION, where the two sides of the partition are required to contain equally many elements. This problem, which we call EQUIPARTITION, is also NP-complete [45, p. 223].

Thus, consider an instance of EQUIPARTITION: "Given positive integers $a_1, a_2, \ldots, a_k$, where k is even; does there exist a vector $\xi \in \{-1,1\}^k$ such that $a^{\mathrm{T}}\xi = 0$ and $\overline{1}^{\mathrm{T}}\xi = 0$?"

Denote $c = \sum_{i=1}^{k} a_i$, and choose

$$W = \begin{pmatrix} 0 & 0 & c+a_1 & c+a_2 & \cdots & c+a_k \\ 0 & 0 & -c-a_1 & -c-a_2 & \cdots & -c-a_k \\ c+a_1 & -c-a_1 & 0 & -1 & \cdots & -1 \\ c+a_2 & -c-a_2 & -1 & 0 & \cdots & -1 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ c+a_k & -c-a_k & -1 & -1 & \cdots & 0 \end{pmatrix}.$$

Clearly W can be constructed from $a_1, a_2, \ldots, a_k$ in polynomial time. Note also that $c \geq k$, and that the vectors $(1,-1,1,1,\ldots,1)$ and $(-1,1,-1,-1,\ldots,-1)$ are always stable vectors. We now prove that the given EquiPartition instance has a solution if and only if this network has at least one more stable vector.

(if) Assume that in the network there is a stable vector of the form $(1, 1, \xi)$ for some $\xi = (\xi_1, \ldots, \xi_k) \in \{-1, 1\}^k$. By rows 1 and 2 of matrix W, we get $c\overline{1}^{\mathrm{T}}\xi + a^{\mathrm{T}}\xi = 0$. Suppose now that $\overline{1}^{\mathrm{T}}\xi \neq 0$. Then $|\overline{1}^{\mathrm{T}}\xi| \geq 2$ (because k is even), and as $|a^{\mathrm{T}}\xi| \leq c$, we see that $|c\overline{1}^{\mathrm{T}}\xi + a^{\mathrm{T}}\xi| \geq c > 0$. Consequently, we must have $\overline{1}^{\mathrm{T}}\xi = 0$, and so also $a^{\mathrm{T}}\xi = 0$.

Let us then prove that the only possible stable vectors besides the vectors $(1, -1, \overline{1})$ and $(-1, 1, -\overline{1})$ are indeed of the form $(1, 1, \xi)$.

1. A stable vector cannot be of the form $(-1, -1, \xi)$, since rows 1 and 2 of matrix W would then require that $0 > c\overline{1}^{\mathrm{T}}\xi + a^{\mathrm{T}}\xi > 0$.

2. If $(1, -1, \xi)$ is a stable vector, then for every $i = 1, 2, \ldots, k$, we have

$$(Wx)_{i+2} = 2c + 2a_i - (\overline{1}^{\mathrm{T}}\xi - \xi_i) \geq 2k + 2 - (k - 1) > 0,$$

   so $(1, -1, 1, \ldots, 1)$ is the only possible stable vector of this form.

3. If $(-1, 1, \xi)$ is a stable vector, then for every $i = 1, 2, \ldots, k$, we have

$$(Wx)_{i+2} = -2c - 2a_i - (\overline{1}^{\mathrm{T}}\xi - \xi_i) \leq -2k - 2 + (k - 1) < 0,$$

   so $(-1, 1, -1, \ldots, -1)$ is the only possible stable vector of this form.

(only if) If ξ is a solution to the given EquiPartition instance, then $(1, 1, \xi)$ is a stable vector. This concludes the proof. □

The result in Theorem 6, except for the zero thresholds, but for two instead of three stable vectors, has also been obtained independently by Lipscomb et al. [46].

We can easily strengthen Theorem 6 to a $\#P$-completeness result after we prove the requisite lemma.

Lemma 2 *The* EQUIPARTITION *problem is #P-complete.*

Proof The problem is clearly in $\#P$. Consider the following transformation from PARTITION, which is $\#P$-complete: Given a PARTITION instance with elements $a_1, a_2, \ldots, a_k$, denote $c = \sum_{i=1}^{n} a_i$, and construct an EQUIPARTITION instance that consists of the elements

$$a_1 + 3c,\ a_2 + 9c,\ \ldots,\ a_k + 3^k c,\ 3c,\ 9c,\ \ldots,\ 3^k c.$$

Let us prove that this is a parsimonious reduction, i.e., that to every PARTITION solution there corresponds exactly one EQUIPARTITION solution, and vice versa.

If y is a solution to the PARTITION instance, then $\sum_{i=1}^{k}(a_i + 3^i c)y_i + \sum_{i=1}^{k} 3^i c(-y_i) = 0$, so $(y, -y)$ is a solution to the EQUIPARTITION instance.

Conversely, let x be a solution to the EQUIPARTITION instance, which means that $\sum_{i=1}^{k} a_i x_i + c\sum_{i=1}^{k} 3^i(x_i + x_{k+i}) = 0$. If $\sum_{i=1}^{k} 3^i(x_i + x_{k+i}) \neq 0$, then

$$|\sum_{i=1}^{k} a_i x_i + c\sum_{i=1}^{k} 3^i(x_i + x_{k+i})| \geq 3c - \sum_{i=1}^{k} a_i > 0.$$

Thus we must have $\sum_{i=1}^{k} 3^i(x_i + x_{k+i}) = 0$. But $|x_i + x_{k+i}| < 3$, so $x_i + x_{k+i} = 0$ for all $i = 1, 2, \ldots, k$. Thus x must be of the form $(y, -y)$ with $y = (x_1, x_2, \ldots, x_k)$, and $\sum_{i=1}^{k} a_i y_i = 0$. □

Corollary 2 *The problem "Given a basic standard Hopfield network; how many stable vectors are there in the network?" is #P-complete.*

Proof The problem is clearly in $\#P$. The reduction used in Theorem 6 provides a parsimonious transformation from EQUIPARTITION to the slightly modified version "Given a basic standard Hopfield network; how many stable vectors *beyond two* are there in the network?". Hence EQUIPARTITION, and any counting problem in $\#P$, can be solved in polynomial time, assuming an ability to solve instances of the original problem in a single step. □

It is known that if in the basic Hopfield network the elements of the connection matrix are independent identically distributed zero-mean Gaussian random variables, the asymptotic estimate for the number of stable bipolar vectors is about $1.05 \times 2^{0.2874n}$ [86, 114].

We now turn to ternary vectors also in the output. For this we need the following lemma about versions of PARTITION and EQUIPARTITION with two nonempty disjoint subsets (of equal sizes) that have equal sums:

- EQUAL SUBSET SUMS is the problem "Given positive integers $a_1, a_2, \ldots, a_k$; does there exist a vector $\xi \in \{-1, 0, 1\}^k$ such that $a^{\mathrm{T}}\xi = 0$?"
- EQUAL EQUISUBSET SUMS is the problem "Given positive integers $a_1, a_2, \ldots, a_k$; does there exist a vector $\xi \in \{-1, 0, 1\}^k$ such that $a^{\mathrm{T}}\xi = 0$ and $\overline{1}^{\mathrm{T}}\xi = 0$?"

Lemma 3 EQUAL SUBSET SUMS *and* EQUAL EQUISUBSET SUMS *are* NP*-complete and* #P*-complete.*

Proof The problems are clearly in *NP* and in *#P*. EQUAL SUBSET SUMS is proven *NP*-complete in [130] with a parsimonious reduction from a version of PARTITION, to which there is a parsimonious reduction from the usual PARTITION problem, which is *#P*-complete. EQUAL EQUISUBSET SUMS can be proven *NP*-complete and *#P*-complete by a reduction from EQUAL SUBSET SUMS equivalent to the reduction in Lemma 2 for proving the *#P*-completeness of EQUIPARTITION. □

We are now ready for the theorem.

Theorem 7 *The problem "Given a basic ternary Hopfield network with tolerance* $\eta \geq 0$*; does the network have at least four stable vectors?" is NP-complete.*

Proof Clearly, the problem is in *NP*. We consider only the case $\eta = 0$, because for $\eta > 0$, we can replace matrix W below by $(\eta + 1)W$. We prove that the problem is *NP*-hard by a reduction from EQUAL EQUISUBSET SUMS. Given an instance of this problem with elements a_1, a_2, $\dots$, a_k, take $c > \sum_{i=1}^{k} a_i + k^2$ and choose

$$W = \begin{pmatrix} 0 & c & a_1+k & a_2+k & \cdots & a_k+k \\ c & 0 & 0 & 0 & \cdots & 0 \\ a_1+k & 0 & 0 & -1 & \cdots & -1 \\ a_2+k & 0 & -1 & 0 & \cdots & -1 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_k+k & 0 & -1 & -1 & \cdots & 0 \end{pmatrix}.$$

Clearly W can be constructed from a_1, a_2, $\dots$, a_k in polynomial time. The vectors $-\overline{1}$, $\overline{0}$, and $\overline{1}$ are always stable vectors, as can be easily verified. We now prove that the given instance of Equal EquiSubset Sums has a solution if and only if this network has at least one more stable vector.

(if) Assume that in the network there is a stable vector of the form $(0,0,\xi)$ for some $\xi = (\xi_1,\dots,\xi_k) \in \{-1,0,1\}^k$, $\xi \neq \overline{0}$. By row 1 of matrix W, we get $a^{\mathrm{T}}\xi + k\overline{1}^{\mathrm{T}}\xi = 0$.

We show now that $\overline{1}^{\mathrm{T}}\xi = 0$. Suppose $\xi_i = 0$ for some i. Then row $i+2$ of W gives the requirement $\xi_i - \overline{1}^{\mathrm{T}}\xi = 0$, i.e., $\overline{1}^{\mathrm{T}}\xi = 0$. Suppose on the other hand that $\xi_i \neq 0$ for all $i = 1, \dots, k$. Then $\xi_i(\xi_i - \overline{1}^{\mathrm{T}}\xi) = 1 - \xi_i\overline{1}^{\mathrm{T}}\xi > 0$ for all $i = 1, \dots, k$. But if $\xi = \overline{1}$ or $\xi = -\overline{1}$, then $1 - \xi_i\overline{1}^{\mathrm{T}}\xi = 1 - k$, so there must be some element ξ_a that is 1 and some element ξ_b that is -1. But then ξ_a gives $1 - \overline{1}^{\mathrm{T}}\xi > 0$ and ξ_b gives $1 + \overline{1}^{\mathrm{T}}\xi > 0$. This is possible only when $\overline{1}^{\mathrm{T}}\xi = 0$.

Now we return to the equality $a^{\mathrm{T}}\xi + k\overline{1}^{\mathrm{T}}\xi = 0$. As $\overline{1}^{\mathrm{T}}\xi = 0$, also $a^{\mathrm{T}}\xi = 0$, so Equal EquiSubset Sums has the solution ξ.

Let us then prove that the only possible stable vectors besides the vectors $-\overline{1}$, $\overline{0}$, and $\overline{1}$ are indeed of the form $(0,0,\xi)$.

1. If $(1,\alpha,\xi)$ with $\alpha \in \{-1,0,1\}$ is a stable vector, then rows 2, $\dots$, $k+2$ of matrix W give $\alpha = 1$ and $\xi = \overline{1}$, but $\overline{1}$ is one of the original stable vectors. The case $(-1,\alpha,\xi)$ is analogous.

2. If $(0,1,\xi)$ is a stable vector, then row 1 of matrix W gives $c + a^{\mathrm{T}}\xi + k\overline{1}^{\mathrm{T}}\xi = 0$, which is impossible. The case $(0,-1,\xi)$ is analogous.

(only if) If $a^{\mathrm{T}}\xi = 0$ and $\overline{1}^{\mathrm{T}}\xi = 0$, then $x = (0, 0, \xi)$ is a stable vector. The stability is easy to check: $(Wx)_{i+2} = \xi_i - \overline{1}^{\mathrm{T}}\xi = \xi_i$. This concludes the proof. $\square$

When the thresholds are zero, the vector $-u$ is stable whenever u is stable, and additionally $\overline{0}$ is stable, so the number of stable vectors is always odd. Consequently, the question "does the network have at least four stable vectors?" is equivalent to the question "does the network have at least five stable vectors?"

Corollary 3 *The problem "Given a basic ternary Hopfield network with tolerance $\eta \geq 0$; how many stable vectors are there in the network?" is #P-complete.*

Proof The problem is clearly in $\#P$. There is a direct correspondence between the solutions for this problem and for EQUAL EQUISUBSET SUMS: $r + 3$ stable vectors correspond to r solutions of EQUAL EQUISUBSET SUMS. Hence EQUAL EQUISUBSET SUMS, and any counting problem in $\#P$, can be solved in polynomial time, assuming an ability to solve instances of the present problem in a single step. $\square$

4.3 Sizes of Attraction Domains

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. As noted in Section 4.1, we restrict our study to synchronous updating. We first give results showing that in general it is difficult to establish whether $\overline{1}$ is a vector that is reachable at all, unless it is given as the input vector. Theorem 8 deals with bipolar networks and Theorem 9 with ternary networks.

Theorem 8 *The problem "Given a basic bipolar Hopfield network; is there any vector going to $\overline{1}$ in one synchronous step?" is NP-complete.*

Proof The problem can be rephrased as "Given a symmetric integer valued matrix W with zero diagonal; is there any bipolar vector x such that $\operatorname{sgn}(Wx) = \overline{1}$?" The problem is obviously in *NP*. Now, given a PARTITION instance with elements $a_1, a_2, \ldots, a_k$, consider

$$W = \begin{pmatrix} 0 & 0 & a_1 & a_2 & \cdots & a_k \\ 0 & 0 & -a_1 & -a_2 & \cdots & -a_k \\ a_1 & -a_1 & 0 & 0 & \cdots & 0 \\ a_2 & -a_2 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_k & -a_k & 0 & 0 & \cdots & 0 \end{pmatrix}.$$

It can be verified that the given instance of PARTITION has a solution if and only if there is an $x \in \{-1, 1\}^{k+2}$ such that $\operatorname{sgn}(Wx) = \overline{1}$. More precisely, each solution ξ to the PARTITION instance corresponds to the present problem's solutions $(1, 1, \xi)$, $(1, -1, \xi)$, and $(-1, -1, \xi)$. $\square$

Counting the number of vectors going to $\overline{1}$ in one step is hard, as we see from the following corollary.

Corollary 4 *The problem "Given a basic bipolar Hopfield network; how many vectors go to $\overline{1}$ in one synchronous step?" is #P-complete.*

Proof The problem is clearly in $\#P$. There is a direct correspondence between the solutions for this problem and for the corresponding PARTITION problem in Theorem 8. Hence PARTITION, and any counting problem in $\#P$, can be solved in polynomial time, assuming an ability to solve instances of the present problem in a single step. □

For ternary standard Hopfield networks, $\overline{1}$ is always reachable, as $\operatorname{sgn}(W\overline{0}) = \overline{1}$. However, this is not the case for $-\overline{1}$. As $\operatorname{sgn}(x) = -1$ if and only if $S_0(x) = -1$, there is a vector $x \in \{-1, 0, 1\}^n$ such that $\operatorname{sgn}(Wx) = -\overline{1}$ if and only if there is a vector x such that $S_0(Wx) = -\overline{1}$. But the existence of such a vector is hard to determine, as we see from the next theorem and from the fact $S_\eta(-x) = -S_\eta(x)$.

Theorem 9 *The problem "Given a basic ternary Hopfield network with tolerance $\eta \geq 0$; is there any vector going to $\overline{1}$ in one synchronous step?" is NP-complete.*

Proof The problem can be rephrased as "Given $\eta \geq 0$ and a symmetric integer valued matrix W with zero diagonal; is there any ternary vector x such that $S_\eta(Wx) = \overline{1}$?" The problem is obviously in *NP*. It is enough to consider the case $\eta = 0$, as we can use matrix $(\eta + 1)W$ instead of W for $\eta > 0$. Now, given an instance of EQUAL EQUISUBSET SUMS with elements a_1, a_2, ..., a_k, and c, take $c > \sum_{i=1}^{k} a_i$ and consider

$$W = \begin{pmatrix} 0 & 1 & 1 & 1 & \cdots & 1 \\ 1 & 0 & 0 & a_1 + c & \cdots & a_k + c \\ 1 & 0 & 0 & -a_1 - c & \cdots & -a_k - c \\ 1 & a_1 + c & -a_1 - c & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & a_k + c & -a_k - c & 0 & \cdots & 0 \end{pmatrix}.$$

We prove that EQUAL EQUISUBSET SUMS has a solution if and only if the present problem has a solution.

(if) Assume that $(Wx)_i > 0$ for all $i = 1, 2, \ldots, k+3$, and denote $x = (\alpha, \xi)$, where $\alpha \in \{-1,0,1\}^3$ and $\xi \in \{-1,0,1\}^k$. From rows 2 and 3 of matrix W we get $-\alpha_1 < a^{\mathrm{T}}\xi + c\overline{1}^{\mathrm{T}}\xi < \alpha_1$. Thus $\alpha_1 = 1$, $a^{\mathrm{T}}\xi = 0$, and $\overline{1}^{\mathrm{T}}\xi = 0$. By row 1 of matrix W, we see that $\alpha_2 + \alpha_3 > 0$ and from rows 4, $\ldots$ $k+3$ of matrix W we see that the combination $(\alpha_2, \alpha_3) = (0, 1)$ is impossible, so $(1,1,1,\xi)$ and $(1,1,0,\xi)$ are the only possible vectors going to $\overline{1}$.

(only if) If $a^{\mathrm{T}}\xi = 0$ and $\overline{1}^{\mathrm{T}}\xi = 0$, then $(1,1,1,\xi)$ and $(1,1,0,\xi)$ are solutions to the present problem. This concludes the proof. □

Corollary 5 *The problem "Given a basic ternary Hopfield network with tolerance $\eta \geq 0$; how many vectors go to $\overline{1}$ in one synchronous step?" is #P-complete.*

Proof The problem is clearly in *#P*. There is a direct correspondence between the solutions for this problem and for the corresponding Equal EquiSubset Sums problem in Theorem 9. Hence Equal EquiSubset Sums, and any counting problem in *#P*, can be solved in polynomial time, assuming an ability to solve instances of the present problem in a single step. □

If $y \in \{-1,1\}^n$ is a stable vector, there is trivially at least one vector $x \in \{-1,1\}^n$ such that $\mathrm{sgn}(Wx) = y$, namely y itself. However, our next theorem shows that, even in this case, asking for another vector in the same attraction domain is *NP*-complete. We first formulate a lemma, of certain interest in itself, showing that a certain very restricted form of 0–1 integer programming is *NP*-complete.

Lemma 4 *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 NP-complete.*

Proof The problem is clearly in *NP*. We prove completeness in two stages: We first reduce the *NP*-complete Minimum Cover problem [45, p. 222] to the present problem

$$\begin{array}{c|c|c|c|}
 & s & k-1 & \\ \hline
r & B & 0 & \begin{matrix}-1\\-1\\ \vdots\\-1\end{matrix} \\ \hline
 & -1\ \ -1\ \ -1 \ldots -1 & 1\ldots 1 & 1 \\ \hline
k-1 & 0 & -I & \begin{matrix}1\\ \vdots \\ 1\end{matrix}\\ \hline
k-1 & 0 & I & \begin{matrix}-1\\ \vdots\\ -1\end{matrix}\\ \hline
\end{array}$$

Figure 5: Matrix C in Lemma 4.

without requiring the matrix to be symmetric or zero diagonal, and then we show how to impose these two conditions.

Consider an instance of MINIMUM COVER: "Given a matrix $B \in \{0,1\}^{r\times s}$ and a constant k; is it possible to select from B a set of $q \leq k$ column vectors $y^{(1)}$, $y^{(2)}$, ..., $y^{(q)}$, so that $\sum_{i=1}^{q} y^{(i)} \geq \overline{1}$?"

For the reduction, construct first from B and k an intermediate matrix C as in Figure 5. We claim that B has a set of $q \leq k$ columns $y^{(1)}, y^{(2)}, \ldots, y^{(q)}$ such that $\sum_{i=1}^{q} y^{(i)} \geq \overline{1}$ if and only if C has a set of columns $x^{(1)}$, $x^{(2)}$, ..., $x^{(p)}$, where $p \geq 1$, such that $\sum_{i=1}^{p} x^{(i)} \geq \overline{0}$. The "only if" direction is clear: Given a set of columns from B, choose from C the corresponding columns plus the k rightmost columns. For the "if" direction, note first that whatever columns are chosen from C, the k rightmost columns, in particular the utter rightmost one, must be among these. This implies that for the columns of B obtained as projections of the choices from C, each of the row sums must be at least 1. Also, there cannot be more than k B-columns, because otherwise the sum on row $r+1$ of C would be negative.

Having constructed the matrix C as above, a symmetric zero diagonal matrix A can then be constructed as in Figure 6. Matrix A satisfies our requirements in the same way

$$
\begin{array}{r|c|c|c|c|}
 & s+k & r+2k-1 & \multicolumn{1}{c}{} & \multicolumn{1}{c}{} \\
\hline
s+k & 0 & C^{\mathrm{T}} & \begin{matrix}0\\0\\\vdots\\0\end{matrix} & \begin{matrix}0\\0\\\vdots\\0\end{matrix} \\
\hline
r+2k-1 & C & 0 & \begin{matrix}-1\\-1\\-1\\-1\\\vdots\\-1\\-1\end{matrix} & \begin{matrix}-1\\-1\\-1\\-1\\\vdots\\-1\\-1\end{matrix} \\
\hline
 & 0\ 0 \ldots\ 0 & -1\ \ -1\ \ -1 \ldots -1 & 0 & -1 \\
\hline
 & 0\ 0 \ldots\ 0 & -1\ \ -1\ \ -1 \ldots -1 & -1 & 0 \\
\hline
\end{array}
$$

Figure 6: Final matrix A in Lemma 4.

as C does, because in fact none of the columns to the right of column $s+k$ of A can ever be included in a set of columns with nonnegative row sums. □

The next theorems say that it is difficult to decide, when using synchronous updating, whether the direct attraction domain, and thereby the general attraction domain, of $\overline{1}$ is bigger than $\{\overline{1}\}$. Theorem 10 deals with bipolar networks and Theorem 11 with ternary networks.

Theorem 10 *The problem "Given a basic bipolar Hopfield network with connection matrix entries from $\{-1,0,1\}$ and with stable vector $\overline{1}$; is there any other vector going to $\overline{1}$ in one synchronous step?" is 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 *NP*. We prove that it is *NP*-hard by a reduction from the problem in Lemma 4. 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 $\operatorname{sgn}(W\overline{1}) = \overline{1}$. 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. □

And, as we see next, counting the number of vectors going to the stable vector $\overline{1}$ in a single synchronous update step is hard.

Corollary 6 *The problem "Given a basic bipolar Hopfield network with connection matrix entries from $\{-1, 0, 1\}$ and with stable vector $\overline{1}$; how many other vectors go to $\overline{1}$ in one synchronous step?" is #P-complete.*

Proof The Minimum Cover problem is $\#P$-complete. To verify this, see the sequence of parsimonious reductions in [45, pp. 48–53, 64] that lead from Satisfiability to Minimum Cover. Furthermore, the reductions given in Lemma 4 and Theorem 10 are parsimonious. □

Turning to ternary standard Hopfield networks, we note again that $\mathrm{sgn}(x) = -1$ if and only if $S_0(x) = -1$, so the next theorem implies that it is hard to determine whether there exists a vector $x \in \{-1, 0, 1\}^n$, $x \neq -\overline{1}$, such that $\mathrm{sgn}(Wx) = -\overline{1}$, when W is a symmetric zero diagonal matrix with entries from $\{-1, 0, 1\}$ and such that $\mathrm{sgn}(W(-\overline{1})) = -\overline{1}$.

Theorem 11 *The problem "Given a basic ternary Hopfield network with tolerance $\eta \geq 0$, with connection matrix entries from $\{-\eta-1, -\eta, \ldots, \eta, \eta+1\}$, and with stable vector $\overline{1}$; is there any other vector going to $\overline{1}$ in one synchronous step?" is NP-complete.*

Proof The problem can be rephrased as "Given $\eta \geq 0$ and a symmetric zero diagonal matrix W with entries from $\{-\eta - 1, -\eta, \ldots, \eta, \eta + 1\}$ such that $S_\eta(W\overline{1}) = \overline{1}$; is there any ternary vector $x \neq \overline{1}$, such that $S_\eta(Wx) = \overline{1}$?" The problem is clearly in *NP*. Again it is enough to consider the case $\eta = 0$. We prove that the present problem is *NP*-hard by a polynomial reduction from the problem in Lemma 4. 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 zero otherwise. Here the subtractions are taken modulo k. The matrix K_1 has 2 elements with value 1 on every row and every column. Denote by $\mathbf{1}$ the matrix whose all entries are 1. Now choose

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

It is easy to check that $S_\eta(W\overline{1}) = \overline{1}$. We prove that there is a $y \in \{0, 1\}^k$, $y \neq \overline{0}$, such that $Ay \geq \overline{0}$ if and only if there is an $x \in \{-1, 0, 1\}^{5k}$, $x \neq \overline{1}$, such that $Wx > \overline{0}$. Let us denote $x = (x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}, x^{(5)})$, where $x^{(i)} \in \{-1, 0, 1\}^k$ for $i = 1, 2, \ldots, 5$.

(if) If $Wx > \overline{0}$ for some $x \neq \overline{1}$, then $x^{(2)} = \overline{1}$ by the third part of matrix W (i.e., rows $2k + 1$, $2k + 2$, $\ldots$, $3k$). From this follows by the second part of W that $x^{(3)} = x^{(4)} = x^{(5)} = \overline{1}$. Moreover, the fourth part of W says that $x^{(1)} \in \{0, 1\}^k$. Now the first part gives

the inequalities $A(\overline{1} - x^{(1)}) + \overline{1} \geq \overline{1}$. Thus $y = \overline{1} - x^{(1)}$ is a solution to $Ay \geq \overline{0}$, and as $x \neq \overline{1}$, also $y \neq \overline{0}$.

(only if) If $Ay \geq \overline{0}$ for some $y \neq \overline{0}$, then $(\overline{1}-y, \overline{1}, \overline{1}, \overline{1}, \overline{1}) \neq \overline{1}$ is a solution to the present problem. This concludes the proof. □

Note that the reduction in Theorem 11 is not preserving the number of solutions, so we do not get any $\#P$-completeness result out of this proof. Nevertheless, we expect the corresponding counting problem to be $\#P$-complete.

4.4 Attraction Radii

Finally we confront the attraction radius for synchronous updating. We show that determining the direct attraction radius is easy, but determining even the two-step attraction radius is *NP*-hard. Unless otherwise stated, we use the 1-metric in this section.

Before turning to the complexity of determining the attraction radii, we state the following theorems (Theorem 12 and 13). The first is by Amari [2] (described also in [71, p. 43]), and is given here for the sake of completeness; the second is a modification of that result to ternary vectors. Amari uses the name *stability number* for the value $s_i(u)$ below.

Theorem 12 *For a synchronous binary Hopfield network, the k-step attraction radius $R_k(u)$ of a stable vector u obeys the recursive inequalities*

$$R_1(u) \geq s_1(u) = \lceil \frac{1}{\omega} h_1(u) \rceil - 1;$$

$$R_k(u) \geq s_k(u) = \max\{\, s_{k-1}(u), \lceil \frac{1}{\omega} h_{s_{k-1}(u)/2+1}(u) \rceil - 1 \,\} \textit{ for } k \geq 2,$$

where ω is the largest $|w_{ij}|$-value and $h_k(u)$ is the k^{th} smallest value of $|\sum_j w_{ij}u_j - \theta_i|$ (and $h_k(u) = h_n(u)$ for $k \geq n$).

Proof We can assume w.l.o.g. that node k receives the k^{th} smallest h-value; i.e., $h_k(u) = |\sum_j w_{kj}u_j - \theta_k|$. The value $h_k(u)$ can be considered as a measure of robustness of node k against perturbations. The idea in the proof is to take a vector x at some distance from u as input vector and show that x converges to u. We consider first the direct attraction radius. Take as x the vector we get from u by changing at most $s_1(u)/2$ elements to their opposite values; thus, the distance between x and u is at most $s_1(u)$. This means that $|\sum_j w_{ij}x_j - \sum_j w_{ij}u_j| \leq \omega s_1(u)$ for all i. But $\omega s_1(u) = \omega(\lceil \frac{1}{\omega} h_1(u) \rceil - 1) < h_1(u)$, so for all i holds $\text{sgn}(\sum_j w_{ij}x_j - \theta_i) = \text{sgn}(\sum_j w_{ij}u_j - \theta_i) = u_i$. Consequently, we get u as result from x in one update step.

We now proceed recursively. Suppose that the claim holds for $k-1$. If the distance between x and u is at most $s_k(u)$, then the case is clear if $s_k(u) = s_{k-1}(u)$; otherwise

$$|\sum_j w_{ij}x_j - \sum_j w_{ij}u_j| \leq \omega s_k(u) \leq \omega(\lceil \frac{1}{\omega} h_{s_{k-1}(u)/2+1}(u) \rceil - 1) < h_{s_{k-1}(u)/2+1}(u),$$

so $\mathrm{sgn}(\sum_j w_{ij}x_j - \theta_i) = u_i$ for $i = s_{k-1}(u)/2 + 1,\ s_{k-1}(u)/2 + 2,\ \ldots,\ n$. In other words, after one update step, the distance to u is at most $s_{k-1}(u)$, from which there is at most $k-1$ update steps to u according to the hypothesis that the claim holds for $k-1$. $\square$

For ternary Hopfield networks we get the following result.

Theorem 13 *For a synchronous ternary Hopfield network with tolerance $\eta \geq 0$, the k-step attraction radius $R_k(u)$ of a stable vector u obeys the recursive inequalities*

$$R_1(u) \geq s_1(u) = \lceil \frac{1}{\omega} h_1(u) \rceil - 1;$$

$$R_k(u) \geq s_k(u) = \max\{\, s_{k-1}(u), \lceil \frac{1}{\omega} h_{\zeta_k(u)}(u) \rceil - 1 \,\} \text{ for } k \geq 2,$$

where ω is the largest $|w_{ij}|$-value, $h_k(u)$ is the k^{th} smallest value of $||\sum_j w_{ij}u_j - \theta_i| - \eta|$ (and $h_k(u) = h_n(u)$ for $k \geq n$), and $\zeta_k(u) = \lfloor s_{k-1}(u)/2 \rfloor + 1$.

Proof We proceed as in the above proof for binary vectors. We can assume w.l.o.g. that node k receives the k^{th} smallest h-value; i.e., $h_k(u) = ||\sum_j w_{kj}u_j - \theta_k| - \eta|$. We take a vector x at some distance from u as input vector and show that x converges to u. We consider first the direct attraction radius. Take as x the vector we get from u by changing r_0 elements to zero and r_1 elements to their opposite values, so that the distance is $r_0 + 2r_1 \leq s_1(u)$. This means that $|\sum_j w_{ij}x_j - \sum_j w_{ij}u_j| \leq \omega s_1(u)$ for all i. But $\omega s_1(u) = \omega(\lceil \frac{1}{\omega} h_1(u) \rceil - 1) < h_1(u)$, so for all i holds $S_\eta(\sum_j w_{ij}x_j - \theta_i) = S_\eta(\sum_j w_{ij}u_j - \theta_i) = u_i$. Consequently, we get u as result from x in one update step.

We now proceed recursively. Suppose that the claim holds for $k-1$. If the distance between x and u is at most $s_k(u)$, then the case is clear if $s_k(u) = s_{k-1}(u)$; otherwise

$$|\sum_j w_{ij}x_j - \sum_j w_{ij}u_j| \leq \omega s_k(u) \leq \omega(\lceil \frac{1}{\omega} h_{\zeta_k(u)}(u) \rceil - 1) < h_{\zeta_k(u)}(u),$$

so $S_\eta(\sum_j w_{ij}x_j - \theta_i) = u_i$ for $i = \lfloor s_{k-1}(u)/2 \rfloor + 1,\ \lfloor s_{k-1}(u)/2 \rfloor + 2,\ \ldots,\ n$. In other words, after one update step, the distance to u is at most $s_{k-1}(u)$, from which there is at most $k-1$ update steps to u according to the hypothesis that the claim holds for $k-1$. $\square$

From these theorems, we get a lower bound on the general attraction radius: As soon as $s_k(u) = s_{k+1}(u)$, all subsequent $s_i(u)$-values are equal to this value, and the attraction radius is at least $s_k(u)$.

Next we show that it is easy to determine the direct attraction radius in a Hopfield network.

Theorem 14 *The problem "Given a synchronous Hopfield network with tolerance $\eta \geq -1$ and with a symmetric integer valued connection matrix with zero diagonal, and also given a stable vector u and a distance k; is the direct attraction radius of u equal to k?" can be determined in polynomial time.*

Proof The bipolar case ($\eta = -1$) is easily obtained by making appropriate changes for the case $\eta = 0$. Assume now that $\eta \geq 0$. Let W be the connection matrix and θ the threshold vector. The $O(n^2)$ time procedure below determines the direct attraction radius of $u = (u_1, u_2, \ldots, u_n)$. Intuitively, this is a greedy algorithm that checks for each node how its inputs must be altered in order to change the state of the node in one update step. The minimum number of changes ($1 \leftrightarrow 0$, $0 \leftrightarrow -1$) equals the distance to the nearest vector that results in something else than u in one update step. If this distance is k, the radius of direct attraction is $k - 1$.

1. `radius` $:= 2n$
2. `for all nodes` i `with` $u_i \in \{-1, +1\}$ `do:`
   (a) `S` $:= []$ `% S is a multiset`
   (b) `for all nodes` j `do:`
      i. `if` $w_{ij} u_i u_j > 0$, `then S` $:=$ `S`$+[|w_{ij}|, |w_{ij}|]$
      ii. `if` $w_{ij} \neq 0$ `and` $u_j = 0$, `then S` $:=$ `S`$+[|w_{ij}|]$
   (c) `sum` $:= |\sum_j w_{ij} u_j - \theta_i|$ `% absolute value of the input to node` i
   (d) `k` $:= 0$
   (e) `while S` $\neq []$ `and sum` $> \eta$ `do:`
      i. `take the largest` $\alpha \in$ `S away from S`
      ii. `sum` $:=$ `sum`$- \alpha$
      iii. `k` $:=$ `k`$+1$
   (f) `if sum` $\leq \eta$ `then radius` $:=$ `min{radius,k`-1`}`
3. `for all nodes` i `with` $u_i = 0$ `do:`
   (a) `S` $:= []$, `T` $:= []$ `% S and T are multisets`

(b) `for all nodes` j `do:`

i. `if` $w_{ij}u_j < 0$, `then S :=` S$+[|w_{ij}|, |w_{ij}|]$

ii. `if` $w_{ij}u_j > 0$, `then T:=` T$+[|w_{ij}|, |w_{ij}|]$

iii. `if` $w_{ij} \neq 0$ `and` $u_j = 0$, `then S :=` S$+[|w_{ij}|]$, `T :=` T$+[|w_{ij}|]$

(c) `sum` $:= \sum_j w_{ij}u_j - \theta_i$ `% this is the input to node` i

(d) `k` $:= 0$

(e) `while S` $\neq []$ `and sum` $\leq \eta$ `do:`

i. `take the largest` $\alpha \in$ `S away from S`

ii. `sum :=` sum$+\alpha$

iii. `k :=` k$+1$

(f) `if sum` $> \eta$ `then radius :=` min$\{$radius,k$-1\}$

(g) `sum` $:= \sum_j w_{ij}u_j - \theta_i$ `% this is the input to node` i

(h) `k` $:= 0$

(i) `while T` $\neq []$ `and sum` $\geq -\eta$ `do:`

i. `take the largest` $\alpha \in$ `T away from T`

ii. `sum :=` sum$-\alpha$

iii. `k :=` k$+1$

(j) `if sum` $< -\eta$ `then radius :=` min$\{$radius,k$-1\}$

4. `return(radius).`

□

Next we consider the problem of computing the general 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 [50].

We study first the situation with bipolar vectors and later extend the results to ternary vectors. For simplicity, we use the Hamming distance in the proof.

Theorem 15 *The problem "Given a synchronous binary Hopfield network with a symmetric integer valued connection matrix with zero diagonal, and also given 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?" 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 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 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 7):

- 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
- 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.

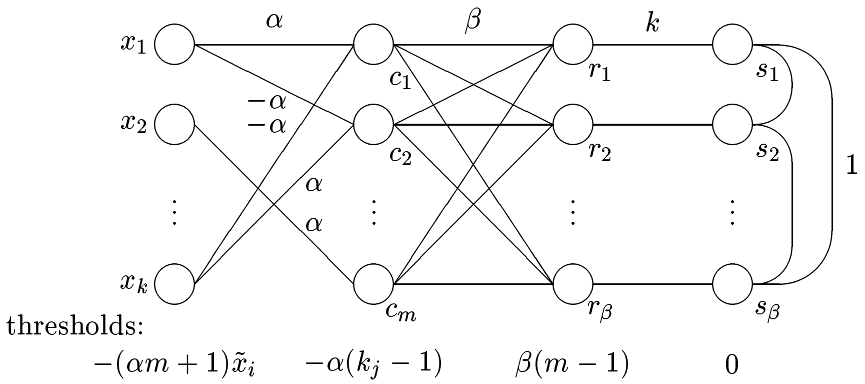

Figure 7: The structure of the Hopfield network in Theorem 15.

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 $\text{sgn}(\pm\alpha m + (\alpha m + 1)\tilde{x}_i) = \tilde{x}_i$. We use here an abbreviation of type $\text{sgn}(a \pm b)$ for $\text{sgn}(x)$, 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 completed. $\square$

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.

Note that the threshold vector was nonzero in the construction. If we add a node with fixed state -1 and put weights from this node equal to the threshold values, we get rid of the nonzero threshold. However, the fixing of the state of this additional node is not possible with any easy method that would preserve the number of steps before convergence.

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 may 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 [97].)

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

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

Now the number of 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 will here use the following stronger construction, which was suggested to us by Viggo Kann [73]. 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 $P = NP$ [57]. Here n is the number of nodes in the graph.

Lemma 5 *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

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

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. □

Theorem 16 *There is no polynomial time algorithm approximating the attraction radii in a synchronous bipolar Hopfield network (with n nodes) within a factor $n^{1-\epsilon}$ for any fixed $\epsilon > 0$, unless $P = NP$.*

Proof We see from the construction of the network in Theorem 15 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 15. 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. Construct from this formula the formula G in the way described in Lemma 5. Now $(-\overline{1}, c_{-\overline{1}}, -\overline{1}, -\overline{1})$ is the stable vector we consider, and the attraction radius is one less than the minimum number of `true` variables needed to satisfy G. But according to Lemma 5, this number cannot be approximated within a factor $n^{1-\epsilon}$ unless $P = NP$; thus also the attraction radius cannot be approximated within that factor. □

Also for ternary Hopfield networks, the attraction radius is hard to determine and to approximate. The metric used is again the 1-metric.

Theorem 17 *The problem "Given a synchronous ternary Hopfield network with tolerance $\eta \geq -1$ and with a symmetric integer valued connection matrix with zero diagonal, and also given a stable vector u and a distance k; is the attraction radius of u less than k?" is NP-hard.*

Proof We can w.l.o.g. concentrate on $\eta = 0$, because multiplying all weights and thresholds by $\eta + 1$, the construction is valid for $\eta > 0$. The metric we use is the 1-metric. The details of the proof are the same as in Theorem 15 except for the following exceptions: We start from the situation in Theorem 16 but add one more variable negated, so that $\overline{1}$ is not satisfying the formula. Now $\tilde{u} = (-\overline{1}, c_{-\overline{1}}, -\overline{1}, -\overline{1})$. We study whether the radius is at most $k-1$. All weights and thresholds are doubled and additionally 1 is subtracted from the thresholds. This ensures that zero is never obtained as a result of an update step. In order to ensure that $\hat{x}$ is at distance at most $k-1$ from $\tilde{x} = -\overline{1}$, we use vector $(\hat{x} - \overline{1})/2$ in the place of $\hat{x}$. Keeping these differences in mind and changing sgn to S_0, we have the desired construction. □

From this follows also the nonapproximability.

Corollary 7 *There is no polynomial time algorithm approximating the attraction radii in a synchronous ternary Hopfield network with tolerance $\eta \geq 0$ (with n nodes) within a factor $n^{1-\epsilon}$ for any fixed $\epsilon > 0$, unless $P = NP$.* □

The construction in Theorem 17 allows the theorem and Corollary 7 to hold also for ternary standard Hopfield networks.

The complexity of determining attraction radii under asynchronous updating is also NP-hard [39].

Komlós and Paturi [78] have studied attraction radii from a probabilistic point of view: They show that for random stored vectors of sufficient length there exists with high probability a positive attraction radius for any stored vector in both synchronous and asynchronous updating.

4.5 Summary of Results

We have shown that given a Hopfield network, it is NP-hard to determine the number of spurious stable vectors in it, or to determine for synchronous updating the size of the direct attraction domain or the general attraction radius of a given stable vector. More strongly, we have shown that most of the counting problems are $\#P$-complete and that the attraction radius is also hard to approximate. This means that it is hopeless to look for exact polynomial time computable formulae for the number of stable vectors or for the attraction radii of stable vectors, since there are no such formulae (if $P \neq NP$). Table 6 shows in which theorem each result is stated.

Table 6: Reference to the number of the theorem in which each of our NP-completeness results is proved for the different Hopfield networks. The possible $\#P$-completeness or non-approximability results can be found directly after the corresponding theorem for the NP-completeness. The abbreviation "t. s." stands for ternary standard.

Problem	bipolar	t. s.	ternary
# stable vectors in the network	6	6	7
# vectors going to given vector	8	9	9
size of attraction domain of stable vector	10	11	11
attraction radius of stable vector	15	17	17

5 Hamming Memories

The *Hamming memory network* goes back to work by Steinbuch [111, 112] and Taylor [116], and was later independently rediscovered by several scientists [14, 28, 82]. The output of the standard Hamming network is determined by the stored vector nearest in Hamming distance (i.e., number of differing bits) to the input vector. This is the reason for the name Hamming memory. We formulate our results using ternary input vectors; consequently, we cannot use the Hamming metric. For simplicity, we still use the name Hamming memory for this network. The motivation for the use of ternary vectors in the Hamming memory is the same as for the Hopfield memory (see Section 3.2): Having zero valued elements instead of wrong elements decreases the distance between the input vector and the desired stored vector. The differences between bipolar and ternary input vectors are rather straightforward: The results given for ternary input vectors are valid for bipolar input vectors after changing $2n$ to n in the formulae.

The standard Hamming memory is optimal in the following respects: All output vectors are stored vectors and the attraction domains are as large as possible. The essential part of the Hamming memory is the winner-take-all (WTA) component, which determines the nearest stored vector by repetitive computations. This is the only part of interest from a computational point of view, so we can consider the study of Hamming networks equivalent to the study of WTA networks.

We study first the standard Hamming memory, which has bipolar output. In Section 5.1 we describe the structure of this model and in Section 5.2 we study the convergence. In Section 5.3 we study the probability that the nearest stored vector to an input vector is unique. After this, we turn to ternary vectors in the output. In Section 5.4 we introduce networks that converge faster than the standard Hamming network and that give vector $\overline{0}$ as output whenever there are several nearest stored vectors and otherwise give the same vector as the standard Hamming network would give. After that we discuss in Chapter 6 a new memory model, which produces ternary output vectors. This new model has no WTA component; it is a feed-forward network.

5.1 Architecture

The Hamming network consists of different layers of nodes: First, values of a function of the distances between the input and the stored vectors are computed, then a WTA network picks the value corresponding to the minimum distance, and finally the corresponding stored vector is given as output. The essential part of the Hamming network is the WTA component, which can be constructed in different ways. The structure studied here is the standard neural realization in accordance with [14, 81, 82], but with zero valued elements allowed in the input.

The Hamming memory is heteroassociative. Throughout this chapter we assume that we have stored m pairs (u^i, v^i), $i = 1, 2, \ldots, m$, of bipolar vectors. The u^i-vectors have length n; the v^i-vectors have length q. The network works as an autoassociative memory when $v^i = u^i$ for all i.

Note that the model can be varied by using different metrics. When the input vectors are bipolar, the Hamming metric and all the p-metrics are related to each other via simple transformations (see Appendix I), but this is not the case for ternary input vectors. The 1-metric corresponds directly to the Hamming metric, and we will use it here.

The network, shown in Figure 8, has the following nodes: n *input nodes*, which can receive states $x_i \in \{-1, 0, 1\}$, p intermediate *distance measuring nodes*, which can receive real-valued states $\hat{y}_i \in [0, 1]$, p intermediate *processing nodes*, which can receive real-valued states $y_i \in [0, 1]$, and q *output nodes*, which can receive states $z_i \in \{-1, 1\}$. The name of a node is just a number; the expressions x_i, y_i, etc., refer to the node states. We use the term *activity level* for the real-valued states of the intermediate nodes.

The connections are as follows: There is a connection with weight a_{ij} from each input node j to each distance measuring node i; there is a connection with weight 1 from each distance measuring node i to the corresponding processing node i; there is a connection with weight w_{ij} from each processing node j to each processing node i, including connections from a node to itself; and there is a connection with weight b_{ij} from each processing node j to each output node i. The capacity of the memory is p. If the memory is not saturated, i.e., the number of stored vector pairs m is strictly less than p, we take the

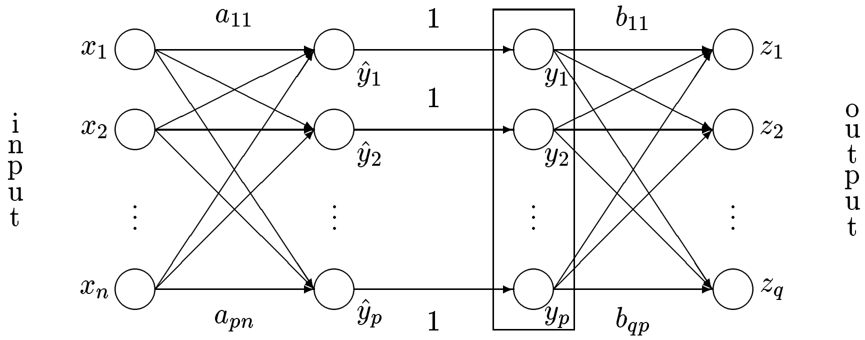

Figure 8: A Hamming network. The nodes in the box are the processing nodes, i.e., the WTA network. The internal connections inside the box are not shown in the figure. The expressions x_1, y_1, etc., denote the states of the nodes. See also Figure 9.

weights in the network as if the rest of the stored vectors pairs were $(\overline{0}, \overline{0})$.

At the beginning of the computation, all nodes other than the input nodes have state zero. The updating is partially synchronous, which means that the states of the nodes within each layer are updated synchronously, but the different levels are dealt with in a specific order (in Figure 8 this order is from left to right).

First, each input node i simply distributes its input element x_i to all distance measuring nodes. Each distance measuring node i computes the *proximity* between stored vector $u^i \in \{-1,1\}^n$ and input vector $x \in \{-1,0,1\}^n$. For this, we choose weights $a_{ij} = u^i_j/(2n)$ and threshold values $-1/2$. The node function is the identity function $id(s) = s$. Thus the proximity value is

$$\hat{y}_i = id\Big(\sum_j a_{ij}x_j + 1/2\Big) = \frac{x^{\mathrm{T}}u^i + n}{2n} = 1 - \frac{d_1(u^i, x)}{2n}. \tag{8}$$

The proximity value is a measure of how close x is to u^i: it is 1 when $x = u^i$, and 0 when $x = -u^i$. In our references (which have bipolar input vectors) the proximity values are not normed to the interval $[0,1]$, but receive values 0, 1, 2, $\ldots$, $2n$. We think it is easier to grasp the computation taking place in the processing nodes when the proximity values are normed to $[0,1]$.

The maximum proximity value is determined in the WTA network: Each processing node i has threshold value 0 and computes iteratively for $t = 1, 2, \ldots$

$$y_i(t) = T\left(\sum_j w_{ij} y_j(t-1)\right), \tag{9}$$

where the node function T is the *threshold logic function*

$$T(y) = \begin{cases} y, & \text{if } y \geq 0 \\ 0, & \text{if } y < 0, \end{cases} \tag{10}$$

and $y_j(0)$ is the activity level processing node j gets from distance measuring node j, i.e., $y_j(0) = \hat{y}_j$. The iteration is repeated until the activity levels of the processing nodes do not change any more. We say that the network has *converged* when such a state is achieved. The WTA network has to be connected in such a way that upon convergence only one processing node is *active*, i.e., has positive activity level.

To get the correct stored output vector as result, we choose weight $b_{ij} = v_i^j$. Each output node i has threshold value 0 and computes

$$z_i = S_0(\sum_j b_{ij}\tilde{y}_j) = S_0(\sum_j v_i^j \tilde{y}_j), \tag{11}$$

where the node function S_0 is the function from formula (2) in Section 3.1, and $\tilde{y}_j$ is the activity level of processing node j at convergence. Note that in this case the input to an output node is always nonzero, so the value of $S_0(0)$ is irrelevant and we can write formula (11) using the usual signum function from formula (1). When finally only one processing node is active, say node k, we have $\tilde{y}_j = 0$ for all $j \neq k$, and thus $z_i = v_i^k$ and the output vector as a whole is v^k. Note that the output vector is always some stored vector.

This network is often called a *unary* network [14, 25], because each intermediate node corresponds to one specific stored vector, and each stored vector is represented by one specific processing node.

For the WTA component of the network, we choose the weights w_{ij} so that for each $i, j = 1, 2, \ldots, p$,

$$\begin{aligned} w_{ii} &= 1 \\ -1/(p-1) \leq \quad w_{ij} &< 0, \quad \text{when } i \neq j. \end{aligned} \tag{12}$$

The restriction (12) on the weights w_{ij} is useful, since all processing nodes can simultaneously become zero if $w_{ij} \leq -1/(p-1)$. To see this, just assume that all activity levels are equal: $y_1(0) = y_2(0) = \cdots = y_p(0) = v$. Then for every i the following holds:

$$y_i(1) = T(v + v\sum_{j \neq i} w_{ij}) \leq T(v - v\frac{p-1}{p-1}) = 0.$$

To ensure convergence, all the weights w_{ij} cannot be equal, since if they were, and the stored vector nearest the input vector were not unique, all the processing nodes corresponding to these nearest stored vectors would remain active forever. This is a problem of some importance, as we will see in the next section, since for random bipolar stored vectors, the probability that there are several stored vectors at the minimum distance from the input vector is in practice not small.

On the other hand, the smaller the weights w_{ij} are, the faster the convergence. Lippman et. al. [82] suggest choosing $w_{ij} = -1/(p-1) + \zeta_{ij}$, where ζ_{ij} is a sufficiently small positive random component. If we now get a situation with several stored vectors at minimum distance from the input, the network will choose one of them.

It is, in fact, enough to have p different weights. Take p different values ϵ_k, where $k = 1, 2, \ldots, p$, so that $0 < \epsilon_k \leq 1/(p-1)$. (There can really be *one* ϵ-value equal to $1/(p-1)$ without problems.) Then for each $i, k = 1, 2, \ldots, p$, where $k \neq i$, take $w_{ik} = -\epsilon_k$. Now formula (9) becomes

$$\begin{aligned} y_i(t) &= T\left(y_i(t-1) - \sum_{k \neq i} \epsilon_k y_k(t-1)\right) \\ &= T\left((1+\epsilon_i)y_i(t-1) - \sum_{k} \epsilon_k y_k(t-1)\right), \end{aligned} \tag{13}$$

and when $y_i(t-1) = y_j(t-1) > 0$, we have $y_i(t) \neq y_j(t)$ or $y_i(t) = y_j(t) = 0$, so nodes with equal activity levels will be separated, unless they both go to zero.

We could now implement the WTA component of the network using fewer connections (see Figure 9). This WTA network has $p+1$ nodes and $3p$ connections instead of the p nodes and p^2 connections in the construct discussed in the preceding text. For the theoretical issues we study in the next section, there is no essential difference between

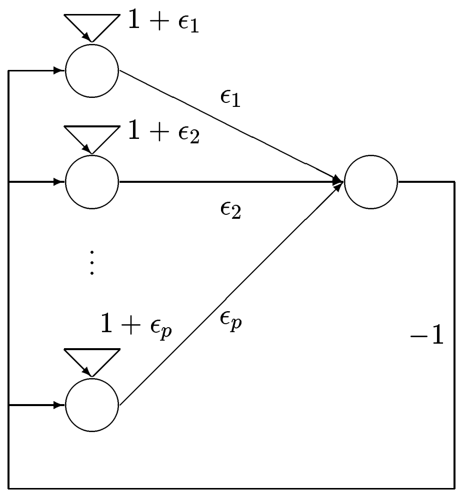

Figure 9: The alternative WTA network construct. The connections from and to other layers are not shown (see Figure 8).

these two structures. The structure in Figure 9 is better for practical implementations, since the memory capacity per connection is higher. Thus it is hardware efficient: Even if the Hamming memory is replicated many times to achieve fault-tolerance, it makes more efficient use of the resources than the Hopfield memory does.

5.2 Convergence

We can now turn to the main question: Does the Hamming network always converge to the correct result, and how fast does it converge? In our references, only the case with equal values w_{ij} is examined; if in the WTA network there is a unique node with the maximum activity level, then the network converges and the activity level of this node is the only one remaining positive. The case with different values w_{ij} is simply passed over ("in practice, ... will still converge and find the maximum" [82, p. 13]).

We assume from now on that we use p different ϵ-values and the (processing) nodes update their values according to formula (13). Naturally, we can assume that $p \geq 2$; otherwise the question is trivial. We start our investigation with some lemmas.

Lemma 6 *Let i and j be processing nodes. Assume that $y_i(t_0) \geq y_j(t_0) > 0$ and $\epsilon_i y_i(t_0) \geq \epsilon_j y_j(t_0)$. Assume also that the values $y_i(t)$ and $y_j(t)$ are strictly positive for all $t = t_0$, $t_0+1, \ldots, t_n$. Then*
(i) $\epsilon_i y_i(t) \geq \epsilon_j y_j(t)$ for all $t = t_0, t_0+1, \ldots, t_n$, and
(ii) $y_i(t) - y_j(t) \geq y_i(t-1) - y_j(t-1)$ for all $t = t_0+1, t_0+2, \ldots, t_n$.

Proof We show first that, whenever (i) holds, (ii) holds as well. For the difference $y_i(t) - y_j(t)$ the terms involving nodes other than i and j cancel as long as $y_i(t)$ and $y_j(t)$ are strictly positive. Using claim (i) we obtain for all $t = t_0+1, t_0+2, \ldots, t_n$ the inequality

$$\begin{aligned} y_i(t) - y_j(t) &= y_i(t-1) - y_j(t-1) + \epsilon_i y_i(t-1) - \epsilon_j y_j(t-1) \\ &\geq y_i(t-1) - y_j(t-1). \end{aligned}$$

Left to prove is claim (i), and we do this by induction over t (up to t_n). The case $t = t_0$ is clear. Assume now that $\epsilon_i y_i(t-1) \geq \epsilon_j y_j(t-1)$, which means that $\epsilon_i/\epsilon_j \geq y_j(t-1)/y_i(t-1)$. As above, it also follows that $y_i(t) - y_j(t) \geq y_i(t-1) - y_j(t-1)$. As the activity level is nonincreasing in each iteration, i.e., $y_i(t-1) \geq y_i(t) > 0$, we get

$$1 - \frac{\epsilon_i}{\epsilon_j} \leq 1 - \frac{y_j(t-1)}{y_i(t-1)} \leq \frac{y_i(t) - y_j(t)}{y_i(t-1)} \leq \frac{y_i(t) - y_j(t)}{y_i(t)} = 1 - \frac{y_j(t)}{y_i(t)},$$

which means that $\epsilon_i y_i(t) \geq \epsilon_j y_j(t)$, and claim (i) is proved. □

Lemma 6 says that if we take any two nodes, as long as the activity levels are strictly positive, the difference in activity levels is nondecreasing in each iteration. The intuitive effect of the requirement $\epsilon_i y_i(t) \geq \epsilon_j y_j(t)$ is that node i "wins" over node j in iteration step t; i.e., if node i had a higher activity level than node j before the iteration, the same will be true after the iteration.

Lemma 7 *Let i and j be processing nodes. Assume that $y_i(t_0) \geq y_j(t_0)$ and $\epsilon_i y_i(t_0) \geq \epsilon_j y_j(t_0)$. Then $y_i(t) \geq y_j(t)$ for all $t \geq t_0$.*

Proof Choose $t \geq t_0$ arbitrarily. Lemma 6 already says that as long as $y_i(t)$ and $y_j(t)$ are strictly positive, $y_i(t) \geq y_j(t)$. The case with $y_i(t) > 0$ and $y_j(t) = 0$ is obvious. Thus we only have to examine the situation with y_i becoming zero when both y_i and y_j were previously positive. By formula (13), we see that $y_i(t) = 0$ means that

$$y_i(t-1) - \epsilon_j y_j(t-1) - \sum_{k \neq i,j} \epsilon_k y_k(t-1) \leq 0,$$

and consequently we get for $y_j(t)$ the threshold logic function's value for

$$\begin{aligned} y_j(t-1) - \epsilon_i y_i(t-1) - \sum_{k \neq i,j} \epsilon_k y_k(t-1) \leq \\ y_j(t-1) - y_i(t-1) + \epsilon_j y_j(t-1) - \epsilon_i y_i(t-1). \end{aligned}$$

But according to Lemma 6, $y_j(t-1) - y_i(t-1) \leq 0$ and $\epsilon_j y_j(t-1) - \epsilon_i y_i(t-1) \leq 0$, so this expression is nonpositive. Hence $y_j(t) = 0$, and the claim is proved. □

Lemma 7 says that the node with highest activity level remains highest all the time, and when the node with second highest activity level has become inactive, all nodes with smaller activity levels have also become inactive. More generally, if we order the nodes by their activity levels, the order will remain unchanged during the iterations.

At last we come to the main point: the convergence theorem. The main observation in the proof is that, regardless of the activity levels of other nodes, the difference between the highest and the second highest activity levels increases until the second highest gets activity level zero.

For the lemmas to hold, we assume that $\epsilon_i y_i(0) \geq \epsilon_j y_j(0)$ for all i and j such that $y_i(0) > y_j(0)$. This is equivalent to saying that $\epsilon_j/\epsilon_j \leq y_i(0)/y_j(0)$ for $y_i(0) > y_j(0)$ (when $y_j(0) \neq 0$). When we construct the network, not knowing what the values $y_k(0)$ will be, this has to be true for arbitrary values. As $y_k(0) \in \{ \frac{i}{2n} \mid i = 0, 1, \ldots, 2n \}$, the least possible ratio $y_i(0)/y_j(0)$ for $y_i(0) > y_j(0)$ is $1/(1 - \frac{1}{2n})$, so the assumption is that $\epsilon_i/\epsilon_j \geq 1 - \frac{1}{2n}$ for all i and j. This means that the ratios ϵ_i/ϵ_j fall into the interval $[1 - \frac{1}{2n}, 1 + \frac{1}{2n-1}]$. When the input vectors are bipolar, the assumption $\epsilon_i/\epsilon_j \geq 1 - 1/n$ for all i and j is enough.

Theorem 18 *Assume for the p different ϵ-values that $0 < \epsilon_i \leq 1/(p-1)$ for all i and $\epsilon_i/\epsilon_j \geq 1 - \frac{1}{2n}$ for all i and j. Then the worst-case convergence time of the WTA component of the Hamming network is*

$$\left\lceil \ln\left(1 + \frac{2}{\sqrt{\frac{\epsilon_i}{\epsilon_j}\frac{y_i(0)}{y_j(0)}} - 1}\right) \Big/ \ln\left(1 + \frac{2}{\frac{1}{\sqrt{\epsilon_i \epsilon_j}} - 1}\right) \right\rceil \tag{14}$$

iterations, where i is the processing node with highest initial activity level and j is the processing node with second highest initial activity level. (If these activity levels are equal, take as higher the activity level of the node with greater ϵ-value.) At convergence only (one of) the processing node(s) with the highest initial activity level remains active.

Proof If there are several processing nodes with highest activity level in the beginning of the WTA computation, then there is a unique node i with highest and a unique node j with second highest activity level after the first iteration. The same holds for the situation with several nodes with second highest activity levels.

Lemma 7 now says that the highest activity level $y_i(t)$ remains highest for all t, and that as soon as the second highest activity level $y_j(t) = 0$, all nodes other than node i are inactive. However, in this case, $y_i(t)$ is still positive, because by Lemma 7 we have $y_i(t-1) \geq y_k(t-1)$ for all k, and when $\epsilon_k < 1/(p-1)$ we get

$$\begin{aligned} y_i(t-1) - \sum_{k \neq i} \epsilon_k y_k(t-1) &> y_i(t-1) - \frac{1}{p-1} \sum_{k \neq i} y_k(t-1) \\ &\geq y_i(t-1)(1 - \frac{p-1}{p-1}) = 0. \end{aligned}$$

Note that $\epsilon_k = 1/(p-1)$ for one k is no problem: Suppose only this node is active besides node i. Then the value of the expression is $y_i(t-1) - y_k(t-1)/(p-1)$, which is strictly positive even for $p = 2$, since $y_i(t-1) > y_k(t-1)$.

Once the other nodes have obtained activity level zero, the activity level of the node remaining active cannot change any more.

Now we have to prove that convergence is really achieved. This is done by noting that $y_i(t)$ decreases with increasing t and that according to Lemma 6 the difference $y_i(t) - y_j(t)$ is nondecreasing. Consequently, the second highest activity level $y_j(t)$ eventually becomes zero.

We prove the convergence by considering the worst possible case; in this way, we obtain a tight bound on the number of iterations. The worst case is the one with $y_i(0) \geq y_j(0) > 0$, and $y_r(0) = 0$ for all $r \neq i, j$, because the difference

$$y_i(t) - y_j(t) = (1 + \epsilon_i)y_i(t-1) - (1 + \epsilon_j)y_j(t-1)$$

does not depend on the activity levels of other nodes, and in this situation the values $y_i(t)$ and $y_j(t)$ decrease in each iteration with the least possible amount. (If $y_r(t-1) > 0$ for some $r \neq i, j$, node r would contribute to the sum $\sum_{k \neq i} y_k(t-1)$ with a positive value, and the activity levels $y_i(t)$ and $y_j(t)$ would decrease with a larger amount.)

We assume here that $\epsilon_i > \epsilon_j$ if $y_i(0) = y_j(0)$, so that node i is the node remaining active at convergence.

The dynamics of the iterations can, as long as $y_j(t) > 0$, be fully expressed by the pair of equations

$$\begin{cases} y_i(t) &= y_i(t-1) - \epsilon_j y_j(t-1) \\ y_j(t) &= y_j(t-1) - \epsilon_i y_i(t-1), \end{cases}$$

and thus convergence is achieved as soon as $y_j(t) \leq 0$ according to these equations. We combine this pair of equations into the single equation

$$y_j(t) = 2y_j(t-1) - (1 - \epsilon_i \epsilon_j)y_j(t-2),$$

which we solve using the technique of generating functions (see, e.g., [101]). Denote

$G(z) = \sum_{t=0}^{\infty} y_j(t) z^t$. We get

$$G(z) = \frac{y_j(0) + z(y_j(1) - 2y_j(0))}{1 - 2z + (1 - \epsilon_i\epsilon_j)z^2} = \frac{A}{1 - \alpha z} + \frac{B}{1 - \beta z},$$

where $A = \frac{1}{2}(y_j(0) - y_i(0)\sqrt{\epsilon_i/\epsilon_j})$, $B = \frac{1}{2}(y_j(0) + y_i(0)\sqrt{\epsilon_i/\epsilon_j})$, $\alpha = 1 + \sqrt{\epsilon_i\epsilon_j}$, and $\beta = 1 - \sqrt{\epsilon_i\epsilon_j}$. Thus $y_j(t) = A\alpha^t + B\beta^t$.

As $\epsilon_i > \epsilon_j$ when $y_i(0) = y_j(0)$ and otherwise $(y_j(0)/y_i(0))^2 < y_j(0)/y_i(0)$, we have $\epsilon_i/\epsilon_j > (y_j(0)/y_i(0))^2$. Thus $A < 0$, and the number of iterations (i.e., the first value t such that $y_j(t) \le 0$) is

$$\left\lceil \frac{\ln(-B/A)}{\ln(\alpha/\beta)} \right\rceil = \left\lceil \ln\left(1 + \frac{2}{\sqrt{\frac{\epsilon_i}{\epsilon_j}}\frac{y_i(0)}{y_j(0)} - 1}\right) \Big/ \ln\left(1 + \frac{2}{\frac{1}{\sqrt{\epsilon_i\epsilon_j}} - 1}\right) \right\rceil.$$

This concludes the proof. □

The assumption that the different ϵ-values do not differ much from one another is necessary, as the following simple example shows.

Example Take $p = 2$, $y_1(0) = 1$, $y_2(0) = 1 - 1/(2n)$, $\epsilon_1 = 1 - 1/n$, and $\epsilon_2 = 1$.

Clearly $\epsilon_1 < \epsilon_2 \le 1/(p-1)$, but $\epsilon_1/\epsilon_2 < 1 - \frac{1}{2n}$, so the assumption in Theorem 18 is not satisfied. Now $y_1(1) = y_2(1) = \frac{1}{2n}$, and $y_1(2) = 0$, but $y_2(2) = \frac{1}{2n^2} > 0$. Thus, even though $y_1(0)$ was higher than $y_2(0)$, the one remaining positive in the end is y_2. □

What does the convergence theorem mean in practice? In the following, we establish in Lemma 8 some conditions that must be satisfied if the ϵ-values are chosen so that the worst-case convergence time is minimized. We call such ϵ-values *optimal*. From these conditions we deduce in Theorem 19 that the worst-case convergence time is $\Omega(p\log(pn))$, and if the nearest stored vector is unique, the worst-case convergence time is $\Omega(p\log n)$. Thereafter, in Theorem 20, we make a specific (near-optimal) choice of ϵ-values and show that these ϵ-values give convergence times $O(p\log(pn))$ and $O(p\log n)$, respectively.

The following lemma describes the properties of an optimal choice of ϵ-values (up to permutations of the values).

Lemma 8 *Assume that* $\epsilon_1 > \epsilon_2 > \cdots > \epsilon_p$. *The optimal* ϵ*-values for the WTA component of the Hamming network satisfy the conditions*

(i) $\epsilon_1 = 1/(p-1)$

(ii) $\epsilon_p = (1 - \frac{1}{2n})/(p-1)$

(iii) $\ln(1 + 2/(\sqrt{\epsilon_i/\epsilon_{i+1}} - 1))/\ln(1 + 2/(1/\sqrt{\epsilon_i\epsilon_{i+1}} - 1)) = c$, *for* $i = 1, 2, \ldots, p-1$, *where* c *is a constant.*

Proof Our objective is to choose ϵ-values so that the maximum convergence time is minimized. From the proof of Theorem 18, we know that the worst possible case has $y_i(0) \geq y_j(0) > 0$, and $y_r(0) = 0$ for all $r \neq i, j$, and that the convergence time in this case is exactly the expression (14) given in the theorem.

Intuitively, when $y_i(0) = y_j(0)$ and $\epsilon_i \geq \epsilon_j \approx 1/(p-1)$, the expression (14) is approximately

$$\frac{p-2}{2} \ln\left(1 + \frac{2}{\sqrt{\epsilon_i/\epsilon_j} - 1}\right).$$

But $\epsilon_i \approx \epsilon_j$, so the value of the expression is very sensitive to the ratio ϵ_i/ϵ_j. The expression (14) is small when ϵ_i/ϵ_j and $\epsilon_i\epsilon_j$ are large.

Suppose the ϵ-values are fixed. Choose i and j arbitrarily. Then the value of the expression (14) depends only on the ratio $y_i(0)/y_j(0)$. The expression (14) is as large as possible, when the ratio $y_i(0)/y_j(0)$ is as small as possible, and thus the worst case is obtained for $y_i(0) = y_j(0)$.

Assume now that $y_i(0) = y_j(0)$. Using the convention in Theorem 18, this implies $\epsilon_i > \epsilon_j$, i.e., $j > i$. Choose $i < p$ arbitrarily. Then $j \in \{i+1, i+2, \ldots, p\}$. We determine the worst possible j. We can suppose that $p > 2$, because the case $p = 2$ is trivial. Denote $\tilde{\mu}_j = \sqrt{\epsilon_i/\epsilon_j} \in]1, (1 - \frac{1}{2n})^{-1/2}] \subseteq]1, 2]$. Then the convergence time is (the nearest integer greater than or equal to)

$$f_i(\tilde{\mu}_j) = \ln\left(1 + \frac{2}{\tilde{\mu}_j - 1}\right) \Big/ \ln\left(1 + \frac{2}{\tilde{\mu}_j/\epsilon_i - 1}\right). \tag{15}$$

As $\epsilon_i < 1$, we have $f_i'(\tilde{\mu}_j) < 0$ for $\tilde{\mu}_j \in]1, 2]$, so $f_i(\tilde{\mu}_j)$ is maximal when $\tilde{\mu}_j$ is minimal, in other words, when $j = i+1$.

Now we can turn to minimizing the convergence time with respect to the ϵ-values. Denote $\mu_i = \sqrt{\epsilon_i/\epsilon_{i+1}}$. Then the convergence time is (the nearest integer greater than or equal to) $\max\{\, c_i \mid i = 1, 2, \ldots, p-1 \,\}$, where

$$c_i = \ln\left(1 + \frac{2}{\mu_i - 1}\right) \Big/ \ln\left(1 + \frac{2}{\mu_i/\epsilon_i - 1}\right). \tag{16}$$

For given μ-values, the value c_i is minimal when value ϵ_i is maximal. Therefore, the optimal choice of ϵ-values has $\epsilon_1 = 1/(p-1)$. (This is condition (i).) Consequently, $1/\epsilon_i = (p-1)\mu_1^2\mu_2^2 \cdots \mu_{i-1}^2$ for $i \geq 2$, and using the convention $\mu_1^2\mu_2^2 \cdots \mu_{i-1}^2 = 1$ for $i = 1$, we can write

$$c_i = \ln\left(1 + \frac{2}{\mu_i - 1}\right) \Big/ \ln\left(1 + \frac{2}{(p-1)\mu_1^2\mu_2^2 \cdots \mu_{i-1}^2\mu_i - 1}\right). \tag{17}$$

Thus c_i depends on μ_1, μ_2, $\ldots$, μ_i. But for given μ_1, μ_2, $\ldots$, μ_{i-1}, the expression (17) for c_i is in fact function $f_i(\mu_i)$ from equation (15), so c_i is minimal when μ_i is maximal. According to the assumption in Theorem 18, $\mu_1^2\mu_2^2 \cdots \mu_{p-1}^2 = \epsilon_1/\epsilon_p \leq 1/(1 - \frac{1}{2n})$, so for maximum μ-values we have $\epsilon_1/\epsilon_p = 1/(1 - \frac{1}{2n})$, from which we get $\epsilon_p = (1 - \frac{1}{2n})/(p-1)$ (condition (ii)).

We now show that the minimum value of $\max\{\, c_i \mid i = 1, 2, \ldots, p-1 \,\}$ is achieved for μ-values such that $c_1 = c_2 = \cdots = c_{p-1}$. Suppose this is not the case. Then the minimum value of $\max\{\, c_i \mid i = 1, 2, \ldots, p-1 \,\}$ is achieved for some choice of μ-values, where there is some maximum value c_a among the c-values. If there are several maximum c-values, choose as c_a the one with largest index. We study separately the cases $a < p-1$ and $a = p-1$.

If $a < p-1$, then $c_i < c_a$ for $i = a+1$, $a+2$, $\ldots$, $p-1$. Now if μ_a is increased and $\mu_{a+1}, \mu_{a+2}, \ldots, \mu_{p-1}$ decreased (so that $\mu_1\mu_2 \cdots \mu_{p-1}$ is a constant), then c_a is decreased and $c_{a+1}, c_{a+2}, \ldots, c_{p-1}$ increased. Thus, we can choose the μ-values so that c_a is decreased and the values c_{a+1}, c_{a+2}, $\ldots$, c_{p-1} are still less than c_a.

If $a = p-1$, we take $b = \max\{\, i \mid c_i < c_a \,\}$; i.e., c_b is the nearest c-value less than c_a. Now if μ_b is decreased and μ_{b+1}, μ_{b+2}, $\ldots$, μ_{p-1} increased (so that $\mu_1\mu_2 \cdots \mu_{p-1}$ is a constant), then c_b is increased and c_{b+1}, c_{b+2}, $\ldots$, c_{p-1} decreased. Thus, we can choose the μ-values so that c_a is decreased and $c_b < c_a$.

When there are several maximum c-values, we can decrease $\max\{ c_i \mid i = 1, 2, \ldots, p-1 \}$ by repeatedly applying the procedure above. But this contradicts the assumption that the given μ-values gave the minimum $\max\{ c_i \mid i = 1, 2, \ldots, p-1 \}$. Thus, in the optimal case, all c-values are equal, which means that for all $i = 1, 2, \ldots, p-1$ we have

$$\ln\left(1 + \frac{2}{\sqrt{\epsilon_i/\epsilon_{i+1}} - 1}\right) \Big/ \ln\left(1 + \frac{2}{1/\sqrt{\epsilon_i \epsilon_{i+1}} - 1}\right) = c,$$

where c is the minimum worst-case convergence time (condition (iii)).

Note that it really is possible to choose the μ-values so that $c_1 = c_2 = \cdots = c_{p-1} = c$. This can be seen as follows. The task is to find $\mu_1, \mu_2, \ldots, \mu_{p-1} > 1$ such that the following $p-1$ conditions are satisfied:

$$c_{i-1} = c_i, \qquad \text{for } i = 2, 3, \ldots, p-1, \tag{18}$$

$$\mu_1 \mu_2 \cdots \mu_{p-1} = (1 - \frac{1}{2n})^{-1/2}. \tag{19}$$

But if we choose a value for μ_{p-1}, the conditions (18) give specific values for μ_{p-2}, μ_{p-3}, $\ldots$, μ_1. Moreover, $\mu_{i-1} \in]1, \mu_i[$, because $\lim_{\mu_{i-1} \to \mu_i} c_{i-1} < c$ and $\lim_{\mu_{i-1} \to 1} c_{i-1} = \infty$, and the greater the value of μ_{p-1} is, the greater are the values of $\mu_1, \mu_2, \ldots, \mu_{p-2}$. Thus there exist continuous strictly increasing functions g_i such that $\mu_i = g_i(\mu_{p-1})$ for $i = 1, 2, \ldots, p-2$. Hence $h(\mu_{p-1}) = \mu_1 \mu_2 \cdots \mu_{p-1} = \mu_{p-1} \prod_{j=1}^{p-2} g_j(\mu_{p-1})$ is a continuous strictly increasing function. As $\lim_{\mu_{p-1} \to 1} h(\mu_{p-1}) = 1$ and $\lim_{\mu_{p-1} \to \infty} h(\mu_{p-1}) = \infty$ (since $h(\mu_{p-1}) > \mu_{p-1}$), there exists some specific value μ_{p-1} for which $h(\mu_{p-1}) = (1 - \frac{1}{2n})^{-1/2}$. This value of μ_{p-1} gives the values of $\mu_1, \mu_2, \ldots, \mu_{p-2}$ satisfying the conditions (18) and (19). $\square$

Using the lemma above, we can state the following theorem.

Theorem 19 *The worst-case convergence time for the WTA component of the Hamming memory is $\Omega(p \log(pn))$ in the general case, where several maximum initial activity levels can occur, and $\Omega(p \log n)$, when there is a unique maximum initial activity level.*

Proof Choose the ϵ-values according to Lemma 8. As in the lemma, denote $\mu_i = \sqrt{\epsilon_i/\epsilon_{i+1}}$. We know from the lemma that $\mu_1 \mu_2 \cdots \mu_{p-1} = \sqrt{\epsilon_1/\epsilon_p} = (1 - \frac{1}{2n})^{-1/2}$, so the least μ-value is at most $(1 - \frac{1}{2n})^{-1/(2p-2)}$, because otherwise all $\mu_i > (1 - \frac{1}{2n})^{-1/(2p-2)}$, and then $\mu_1 \mu_2 \cdots \mu_{p-1} > (1 - \frac{1}{2n})^{-1/2}$. In the same way, the largest μ-value is at least $(1 - \frac{1}{2n})^{-1/(2p-2)}$.

The worst cases are those with $y_i(0) = y_{i+1}(0) > 0$, and $y_k(0) = 0$ for $k \neq i, i+1$, and the convergence time given by expression (14) is the same for all $i = 1, 2, \ldots, p-1$.

We study now the case where μ_i is the least μ-value. From the series expansion $(1+x)^r = 1 + \binom{r}{1}x + \binom{r}{2}x^2 + \cdots$, we see that

$$\mu_i \leq \left(1 - \frac{1}{2n}\right)^{-1/(2p-2)} = \left(1 + \frac{1}{2n-1}\right)^{1/(2p-2)} < 1 + \frac{1}{2(p-1)(2n-1)},$$

so

$$\ln\left(1 + \frac{2}{\mu_i - 1}\right) > \ln(1 + 2(p-1)(2n-1)) = \Omega(\log(pn)).$$

Clearly $\sqrt{\epsilon_i \epsilon_{i+1}} < 1/(p-1)$, so $1/\sqrt{\epsilon_i \epsilon_{i+1}} - 1 > p - 2$. As $\ln(1+x) < x$ for $x > 0$, we see that

$$\left(\ln\left(1 + \frac{2}{1/\sqrt{\epsilon_{p-1}\epsilon_p} - 1}\right)\right)^{-1} > \frac{p-2}{2}.$$

Thus the convergence time is $\Omega(p \log(pn))$.

We now turn to cases with a unique maximum initial activity level. Suppose u_j is the largest μ-value. The worst case has then $y_j(0) = 1 - \frac{1}{2n}$, $y_{j+1}(0) = 1$, and $y_k(0) = 0$ for $k \neq j, j+1$ (see Lemma 8). Then

$$\sqrt{\frac{\epsilon_{j+1}}{\epsilon_j} \frac{y_{j+1}(0)}{y_j(0)}} \leq \left(1 - \frac{1}{2n}\right)^{1/(2p-2)} \left(1 - \frac{1}{2n}\right)^{-1} < \left(1 - \frac{1}{2n}\right)^{-1} = 1 + \frac{1}{2n-1}.$$

Thus

$$\ln\left(1 + \frac{2}{\sqrt{\epsilon_{j+1}/\epsilon_j y_{j+1}(0)/y_j(0)} - 1}\right) > \ln(1 + 2(2n-1)) = \Omega(\log n),$$

and the denominator is the same as in the general case. The claim for a unique maximum initial activity level follows. □

As explicit expressions for the optimal ϵ-values are difficult to derive from the conditions in Lemma 8, we will in the following describe a near-optimal choice of ϵ-values and show that the worst-case convergence time using these ϵ-values is $O(p \log(pn))$, in the general case, and $O(p \log n)$, when there is a unique maximum initial activity level.

Theorem 20 *Denote* $\mu = (1 - \frac{1}{2n})^{-1/(2p-2)}$ *and* $\nu = \mu^{2p-3}$*. The worst-case convergence time for the WTA component of the Hamming memory with* $\epsilon_i = \frac{1}{p-1}(1 - \frac{1}{2n})^{(i-1)/(p-1)}$ *for* $i = 1, 2, \ldots, p$ *is*
(i) in the general case, where several maximum initial activity levels can occur,

$$\left\lceil \ln\left(1 + \frac{2}{\mu - 1}\right) \Big/ \ln\left(1 + \frac{2}{(p-1)\nu - 1}\right) \right\rceil = O(p \log(pn)),$$

(ii) when there is a unique maximum initial activity level,

$$\left\lceil \ln\left(1 + \frac{2}{\nu - 1}\right) \Big/ \ln\left(1 + \frac{2}{(p-1)\nu - 1}\right) \right\rceil = O(p \log n).$$

Proof The ϵ-values in this theorem results from putting $\epsilon_1/\epsilon_2 = \epsilon_2/\epsilon_3 = \cdots = \epsilon_{p-1}/\epsilon_p$, while still $\epsilon_1 = 1/(p-1)$ and $\epsilon_p = (1 - \frac{1}{2n})/(p-1)$.

For fixed p and n, the ratio $\mu = \sqrt{\epsilon_i/\epsilon_{i+1}} = (1 - \frac{1}{2n})^{-1/(2p-2)} > 1$ is a constant, so from the expression (16) for the worst-case convergence times c_i, we see that the maximum c_i is the one with least value ϵ_i. Thus $i = p - 1$ is the worst case. Now $1/\sqrt{\epsilon_{p-1}\epsilon_p} = (p-1)\mu^{2p-3} = (p-1)\nu$, so the worst-case convergence time is

$$\left\lceil \ln\left(1 + \frac{2}{\mu - 1}\right) \Big/ \ln\left(1 + \frac{2}{(p-1)\nu - 1}\right) \right\rceil.$$

The worst case for $y_i(0) > y_j(0)$ is the case $y_{p-1}(0) = 1 - \frac{1}{2n}$, $y_p(0) = 1$, and $y_k(0) = 0$ for $k = 1, 2, \ldots, p-2$. Then $\sqrt{\epsilon_p/\epsilon_{p-1}} y_p(0)/y_{p-1}(0) = \mu^{-1}\mu^{2p-2} = \mu^{2p-3} = \nu$, so when there is a unique maximum initial activity level, the worst-case convergence time is

$$\left\lceil \ln\left(1 + \frac{2}{\nu - 1}\right) \Big/ \ln\left(1 + \frac{2}{(p-1)\nu - 1}\right) \right\rceil.$$

From the series expansion of $(1+x)^r$, we get for $p \geq 2$ the inequalities

$$\begin{aligned} \mu &= \left(1 - \frac{1}{2n}\right)^{-1/(2p-2)} > 1 + \frac{1}{4(p-1)n}, \\ \nu &= \left(1 - \frac{1}{2n}\right)^{-(2p-3)/(2p-2)} > 1 + \frac{2p-3}{4(p-1)n} \geq 1 + \frac{1}{4n}. \end{aligned}$$

Consequently,

$$\ln\left(1 + \frac{2}{\mu - 1}\right) < \ln(1 + 8(p-1)n) = O(\log(pn)),$$

and

$$\ln\left(1 + \frac{2}{\nu - 1}\right) < \ln(1 + 8n) = O(\log n).$$

For $n \geq 2$ we have

$$\nu = \left(1 - \frac{1}{2n}\right)^{-(2p-3)/(2p-2)} < 1 + \frac{1}{2n-1} \leq \frac{4}{3},$$

so

$$\ln\left(1 + \frac{2}{(p-1)\nu - 1}\right) > \ln\left(1 + \frac{6}{4p-7}\right).$$

But $\ln(1+x) > x - x^2/2$ for $-1 < x \leq 1$, so assuming $p \geq 7$ we get

$$\ln\left(1 + \frac{6}{4p-7}\right) > \frac{6}{4p-7} - \left(\frac{6}{4p-7}\right)^2 > \frac{1}{p-2}.$$

Thus

$$\left(\ln\left(1 + \frac{2}{(p-1)\nu - 1}\right)\right)^{-1} < p - 2 = O(p),$$

and the claims follow. □

Note that combining Theorems 19 and 20 we get as result that the worst-case convergence time for the Hamming memory is $\Theta(p \log(pn))$ in the general case and $\Theta(p \log n)$, when there is a unique nearest stored vector.

If the input vectors are bipolar, the optimal value for ϵ_p in Lemma 8 is $(1 - \frac{1}{n})/(p-1)$, the corresponding near-optimal ϵ-values in Theorem 20 are $\epsilon_i = \frac{1}{p-1}(1 - \frac{1}{n})^{(i-1)/(p-1)}$, and the worst-case convergence times are still $\Theta(p \log(pn))$ and $\Theta(p \log n)$, respectively.

The worst-case convergence times with $\epsilon_i = \frac{1}{p-1}(1 - \frac{1}{2n})^{(i-1)/(p-1)}$ are plotted in Figure 10 for $p = 100$ and $1 \leq n \leq 1000$. There is a drop in the beginning of the plot for the worst-case convergence time in the general case, because the time bound decreases in the beginning and achieves its minimum for $n = 4$. We see that the worst-case convergence time when there is a unique nearest stored vector is about half the worst-case convergence time in the general case. Note that if the input vectors are bipolar, we can change $2n$ to n in Theorem 20. Figure 10 can thus be used also for bipolar input vectors by looking at the values for $n/2$; i.e., the worst-case convergence time for $n = 1000$ can be read out from Figure 10 at point $n = 500$.

The optimal sequential time complexity for finding the maximum in a collection of p values is $\Theta(p)$. This holds not only for the worst case, but also for the average case.

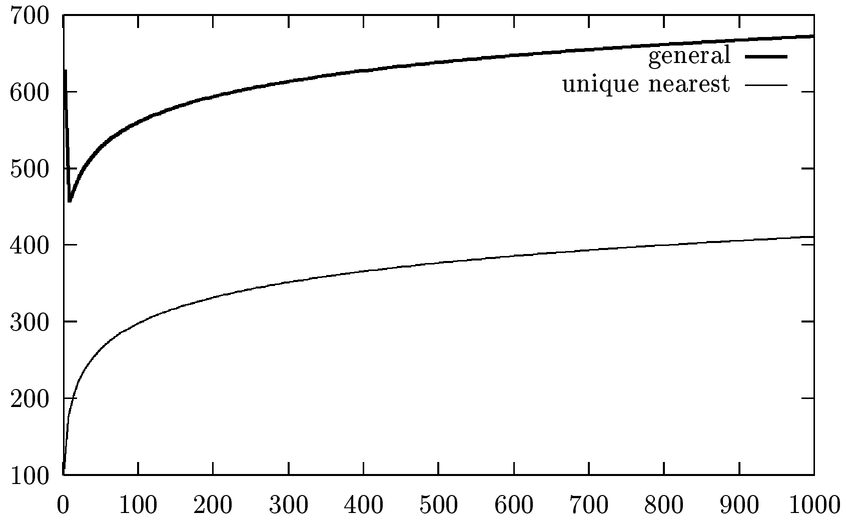

Figure 10: Worst-case convergence times of the Hamming memory with $\epsilon_i = \frac{1}{p-1}(1-\frac{1}{2n})^{(i-1)/(p-1)}$ in the general case and when there is a unique nearest stored vector for $p = 100$ and $1 \leq n \leq 1000$.

However, for the Hamming network, the average-case behavior is much better than the worst-case behavior.

We made some simulations to illustrate the average-case convergence times vs. the worst-case convergence times in the Hamming memory with $\epsilon_i = \frac{1}{p-1}(1-\frac{1}{n})^{(i-1)/(p-1)}$. The initial activity levels were produced assuming that the p stored vectors were independent random bipolar vectors of length n and the input vector was a random bipolar vector independent of the stored vectors. This means that the proximity values are binomially distributed: The probability of the proximity between the input vector and a stored vector being i/n is $\binom{n}{i}/2^n$.

The results are shown in Table 7. The values in the table are means over 1000 ($p = 10$, 100) or 100 ($p = 1000$) input vectors. The mean convergence times seem to increase linearly with p and, contrary to the worst-case times, to decrease with n. This is because the larger n is, the more common is the situation with a unique nearest stored vector. This is explored in more detail in the next section. If we study only instances with unique

Table 7: Convergence times of the standard Hamming memory with $\epsilon_i = \frac{1}{p-1}(1 - \frac{1}{n})^{(i-1)/(p-1)}$, when the input vectors are bipolar. Each entry in the table consists of the mean obtained by the simulations and (in parenthesis) the worst-case bound calculated using the formula in Theorem 20 (i).

$p \setminus n$	10	100	1000
10	11.52 (29)	10.66 (38)	9.64 (48)
100	152.5 (453)	134.1 (530)	119.8 (639)
1000	1380 (5852)	1515 (6506)	1274 (7601)

Table 8: Convergence times of the standard Hamming memory with $\epsilon_i = \frac{1}{p-1}(1 - \frac{1}{n})^{(i-1)/(p-1)}$, when the input vectors are bipolar and there is always a unique nearest stored vector. Each entry in the table consists of the mean obtained by the simulations and (in parenthesis) the worst-case bound calculated using the formula in Theorem 20 (ii).

$p \setminus n$	10	100	1000
10	6.34 (15)	7.67 (25)	8.64 (35)
100	51.4 (163)	81.1 (265)	93.5 (377)
1000	303 (1635)	713 (2671)	952 (3801)

nearest stored vectors, see Table 8, the mean convergence times increase with n, and still increase linearly with p.

As can be seen from the tables, the mean convergence times can even be less than p, while sequential computation always needs p steps. The Hamming memory is much faster than the asynchronous Hopfield memory: When 20 random bipolar input vectors of length 200 were stored and 1000 random bipolar vectors were given as input (the same vectors as in Table 1 in Section 3.3), the mean of the convergence times for (the WTA component of) the standard Hamming memory with $\epsilon_i = \frac{1}{p-1}(1-\frac{1}{n})^{(i-1)/(p-1)}$ was 22, and the maximum convergence time was 84, while the mean number of changing update steps in the asynchronous standard Hopfield network was 146 (see Table 1 in Section 3.3), and even the minimum number of changing update steps was 87. As can be seen from Table 3 in Section 3.3, the synchronous Hopfield memory is faster than the Hamming memory, but it results rather often in cycles of length two. Therefore, the outcome of the asynchronous Hopfield memory is better comparable to the outcome of the Hamming memory.

5.3 Probability of a Unique Nearest Stored Vector

We see from Theorem 20 that the problematic instances are those with several nearest stored vectors — if the ϵ-values were all equal, the computation would not even converge. An interesting question is, therefore, If the stored vectors and the input vector are randomly chosen, how high is the probability that there is a unique nearest stored vector to the input vector? The following calculation shows that this probability is high when the vector length is high. This means that the problematic instances are rare when the vectors are long. However, for short vectors, the probability is not so high, so the situation with several nearest stored vectors must be taken care of efficiently: The ϵ-values cannot be equal.

Let us study the probability mentioned above, when all vectors are bipolar vectors of length n. Assume that the elements of the m stored vectors and the input vector are randomly chosen to be -1 or $+1$ with equal probability independently of other elements. This means that we allow the same vector to be stored several times. Clearly, we can assume that there are at least two stored vectors; otherwise there is no analysis to perform. Thus, the formulae below are valid for $m \geq 2$ and $n \geq 1$. Recall that $\tilde{y}_i$ is the proximity between stored vector u^i and input vector x; i.e., $\tilde{y}_i = (x^{\mathrm{T}} u^i + n)/(2n)$. We denote the probability of event A by $\Pr(A)$. Obviously $\Pr(\tilde{y}_i = s/n) = \binom{n}{s}/2^n$. We now study the probability that there is a unique nearest stored vector to the input vector, and denote this probability $U(m,n)$. The probability of uniqueness is

$$
\begin{aligned}
U(m,n) &= \sum_{s=1}^{n} \Pr(\text{the highest } \tilde{y}_i = s/n, \text{ and } \tilde{y}_j < s/n \text{ for all } j \neq i) \\
&= \sum_{s=1}^{n} [\Pr(\tilde{y}_1 = s/n \text{ and } \tilde{y}_2 < s/n \text{ and } \tilde{y}_3 < s/n \text{ and } \ldots \text{ and } \tilde{y}_m < s/n) \\
&\quad + \Pr(\tilde{y}_1 < s/n \text{ and } \tilde{y}_2 = s/n \text{ and } \tilde{y}_3 < s/n \text{ and } \ldots \text{ and } \tilde{y}_m < s/n) + \cdots \\
&\quad + \Pr(\tilde{y}_1 < s/n \text{ and } \ldots \text{ and } \tilde{y}_{m-2} < s/n \text{ and } \tilde{y}_{m-1} < s/n \text{ and } \tilde{y}_m = s/n)] \\
&= m \sum_{s=1}^{n} \Pr(\tilde{y} = s/n)\, [\Pr(\tilde{y} < s/n)]^{m-1} \\
&= m \sum_{s=1}^{n} \Pr(\tilde{y} = s/n) \left[\sum_{t=0}^{s-1} \Pr(\tilde{y} = t/n) \right]^{m-1}
\end{aligned}
$$

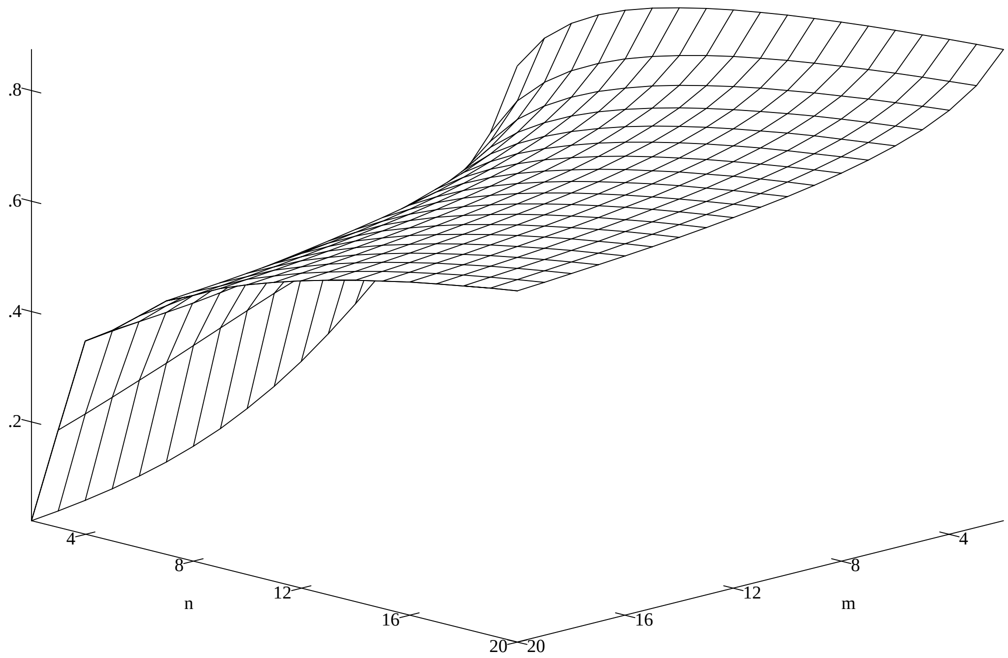

Figure 11: The probability $U(m,n)$.

$$= \frac{m}{2^{mn}} \sum_{s=1}^{n} \left[\binom{n}{s} \left[\sum_{t=0}^{s-1} \binom{n}{t} \right]^{m-1} \right]. \tag{20}$$

The probability $U(m,n)$ for small m and n is plotted in Figure 11. For fixed n, the probability behaves rather interestingly (see Figure 12). The waves occurring at large values of m can be explained in the following way. If j is quite close to n, we can find a value for m such that in most cases at least two of the m random values are equal to j, but no one of them equals $j+1$. This situation corresponds to a local minimum of the probability $U(m,n)$. When m grows, it becomes reasonably likely that one random variable attains the value $j+1$ but all the other values are strictly smaller; this gives rise to a local maximum. Technically, every maximum of $U(m,n)$ corresponds to the maximum of a single term in the expression (20).

In general, the probability that there are q nearest stored vectors, excluding the case

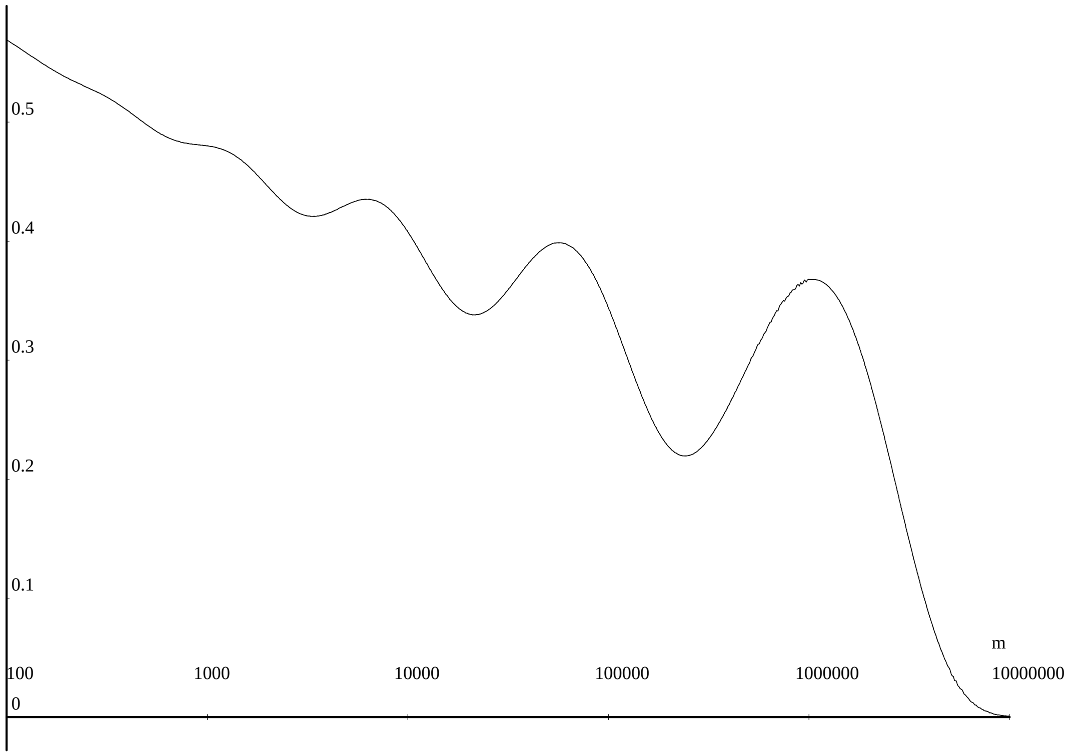

Figure 12: The probability of uniqueness for $n = 20$. Note that the ordinate axis has logarithmic scale.

with all proximity values being 0, is

$$\frac{1}{2^{mn}}\binom{m}{q}\sum_{s=1}^{n}\left[\binom{n}{s}^{q}\left[\sum_{t=0}^{s-1}\binom{n}{t}\right]^{m-q}\right]. \tag{21}$$

(Taking into account that the probability for all proximity values being 0 is $1/2^{mn}$, we get the formidable equality

$$1+\sum_{q=1}^{m}\binom{m}{q}\sum_{s=1}^{n}\left[\binom{n}{s}^{q}\left[\sum_{t=0}^{s-1}\binom{n}{t}\right]^{m-q}\right] = 2^{mn}.)$$

We can extend this discussion to ternary input vectors as follows: Suppose that we put r elements to zero and choose the remaining $n - r$ elements randomly to be -1 or $+1$ with equal probability. Then the proximity value is $(2s + r)/(2n)$ with probability

Table 9: Probabilities $U(m,n)$ for some values of m and n.

$m \setminus n$	10	50	100
10	.607	.803	.857
50	.474	.729	.800
100	.441	.704	.780
1000	.368	.635	.726
2500	.212	.612	.706
10000	.001	.576	.671

$\binom{n-r}{s}/2^{n-r}$. Consequently, the probability that there is a unique nearest stored vector to the input vector is $U(m, n-r)$.

It is impossible to give expression (20) in a closed form without binomial sums, so we study the limit of this probability when m is a function of n and n grows. Clearly, $\lim_{n\to\infty} U(m,n) = 1$ for fixed m and $\lim_{m\to\infty} U(m,n) = 0$ for fixed n, so there is some boundary on the functions $m(n)$ giving limit probability 1. In Appendix II we show that $\lim_{n\to\infty} U(m(n),n) = 1$ for all *subexponential* functions $m(n)$, i.e., when $\log m(n) = o(n)$. In particular, all polynomials are subexponential functions. We can formulate this result as a theorem:

Theorem 21 *Assume that the stored vectors are random vectors of length n. Then the probability that a random input vector has a unique nearest stored vector approaches 1 as n grows, whenever the number of stored vectors is subexponential in n.* $\square$

Probability values for some m and n are given in Table 9. The table shows that despite the probability value in the limit, the probabilities are *not* so high that we in practice could neglect the possibility of having several stored vectors at the minimum distance from the input vector.

In the next section we turn to the question of how we can speed up the convergence, and in Chapter 6 we ultimately even get rid of the iterations completely.

5.4 Zero Valued Elements and Faster Convergence

In the previous section we saw that the situations with several nearest stored vectors were problematic. A natural way to deal with such a situation is to give a "don't know" answer as result. Taking all ϵ-values equal in iteration (13) would result in nonconvergence. However, since we know from Theorem 18 that the maximum convergence time in situations with a unique nearest stored vector is

$$\left\lceil \ln(4n-1) \Big/ \ln(1+2/(\epsilon^{-1}-1)) \right\rceil, \tag{22}$$

we could perform at most this number of iterations and then stop. The output vector received through formula (11) would then correspond to (the signs of) a linear combination of the nearest stored vectors.

The following lemma describes how an iteration with equal ϵ-values can be changed without affecting the outcome of the iterations.

Lemma 9 *The iterations*

$$y_i(t) = T\left(y_i(t-1) - \epsilon \sum_{k \neq i} y_k(t-1)\right) \tag{23}$$

and

$$y_i(t) = T\left(y_i(t-1) - \frac{\epsilon}{1+\epsilon} \sum_{k} y_k(t-1)\right) \tag{24}$$

converge in the same number of iterations to the same solution (i.e., the same node remains active, if any).

Proof Clearly,

$$\begin{aligned} y_i(t-1) - \epsilon \sum_{k \neq i} y_k(t-1) &= (1+\epsilon)\, y_i(t-1) - \epsilon \sum_{k} y_k(t-1) \\ &= (1+\epsilon) \left[y_i(t-1) - \frac{\epsilon}{1+\epsilon} \sum_{k} y_k(t-1) \right]. \end{aligned}$$

Consequently, the iterations (23) and (24) have the same nodes active in each iterations, from which the claim follows. □

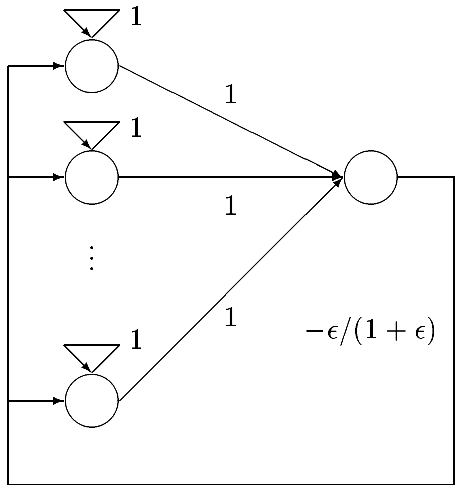

Figure 13: WTA network with equal ϵ-values. The connections from and to other layers are not shown (cf. Figure 9).

Especially, the iterations

$$y_i(t) = T\left(y_i(t-1) - \frac{1}{p-1}\sum_{k\neq i} y_k(t-1)\right) \tag{25}$$

(i.e., iteration (13) with $\epsilon = 1/(p-1)$) and

$$y_i(t) = T\left(y_i(t-1) - \frac{1}{p}\sum_{k} y_k(t-1)\right) \tag{26}$$

are computationally equivalent.

The actual activity levels are different in the iterations (23) and (24). Formula (24) gives rise to smaller activity levels, so if there is a risk of running into problems with the accuracy of the activity levels, formula (23) is better to use. The WTA component of the network could now be constructed as in Figure 13.

However, we can receive a "don't know" answer in fewer iterations than above and at the same time obtain a general speed-up by changing the connection weights during the iterations. Assume that the ϵ-values are equal at each iteration. Denote the ϵ-value at iteration t by $\epsilon(t)$. Now if $\epsilon(t)$ increases with time, the situation with several nearest stored vectors will eventually force the processing nodes to become inactive. We say that the iterative computation has converged when *at most* one node remains active. Consequently, the computation always converges.

Now, according to Lemma 9, the following iterations are computationally equivalent:

$$y_i(t) = T\left(y_i(t-1) - \epsilon(t)\sum_{k\neq i} y_k(t-1)\right) \tag{27}$$

and

$$y_i(t) = T\left(y_i(t-1) - \frac{\epsilon(t)}{1+\epsilon(t)}\sum_{k} y_k(t-1)\right), \tag{28}$$

so the WTA network can be constructed as in Figure 13 with only one connection weight changing at each iteration.

Note that the choice $\epsilon(t) = 1/(p-t)$ means that iteration (27) is computationally equivalent to iteration

$$y_i(t) = T\left(y_i(t-1) - \epsilon(t-1)\sum_{k} y_k(t-1)\right). \tag{29}$$

The computation using iteration (29) has to stop when convergence is achieved, because the next iteration would cause even the last active node to become inactive.

The next theorem characterizes the behavior of the network with changing connection weights: The result is vector $\overline{0}$ if the nearest stored vector is not unique; otherwise the result is the (unique) nearest stored vector. If the number of stored vector pairs m is less than p, the missing activity levels can simply be considered to have value zero.

Theorem 22 *Let $a(t)$ be the number of active nodes at the beginning of iteration $t \geq 1$. Assume that $1/(p-t) \leq \epsilon(t) \leq 1/(a(t)-1)$ for $1 \leq t < p$. Then the iterative process determined by formula (27) converges in at most $p-1$ iterations. If there is a unique node with the maximum initial activity level, then this node is the only one that remains active at convergence; otherwise all nodes are inactive at convergence.*

Proof We show by induction over t that at the end of iteration t at most $p-t$ nodes are active. Since the end of iteration $t-1$ is the starting point of iteration t, we regard the initial situation as the end of iteration number zero. This case is clear: At most all nodes are active.

Assume now that the claim holds for iteration $t-1$ and study iteration number t. As a node that has reached activity level zero cannot become active any more, there are clearly at most $p-t$ active nodes after the iteration if there were at most $p-t$ active nodes before the iteration. Left to examine is the case with $p-t+1$ active nodes before the iteration. In this case $\epsilon(t)$ must be exactly $1/(p-t)$. Now $\epsilon(t)/(1+\epsilon(t)) = 1/(p-t+1)$, so according to Lemma 9, the iteration number t corresponds now to the iteration $y_i(t) = T(y_i(t-1) - \frac{1}{p-t+1}\sum_k y_k(t-1))$, i.e., the mean of the positive $y_k(t-1)$-values are subtracted from each activity level. But there is always at least one value that is less than or equal to the mean. This value becomes zero as a result of the iteration; thus there are at most $p-t$ active nodes left after the iteration, and the claim is proved.

This means that after iteration $p-1$ at most one node is active; i.e., the computation has converged.

Assume that there is a unique node with highest initial activity level. Let this node be node i. We apply again Lemma 9 and use the iteration of form (28). At each iteration, all nodes that remain active have their activity levels decreased by the same amount. Hence, if $y_i(t)$ remains positive, it is highest among the $y_k(t)$-values. As $\frac{\epsilon(t)}{1+\epsilon(t)} \leq \frac{1}{a(t)}$, the expression $\frac{\epsilon(t)}{1+\epsilon(t)}\sum_k y_k(t-1)$ is at most the mean of the positive $y_k(t-1)$-values, and as $y_i(t-1)$ is higher than this mean, $y_i(t)$ must remain positive. Consequently, if there is a unique node with highest initial activity level, this node is the one that remains active at convergence.

In the same way, if there are $r > 1$ nodes with the highest initial activity level, the computation proceeds as above until only these nodes remain active and $\epsilon(t) = 1/(r-1)$. All these nodes have equal activity levels. Consequently, they all become inactive in the next iteration. $\square$

Note that, by the preceding proof, the assumptions in the theorem are satisfied by the choice $\epsilon(t) = 1/(p-t)$. In this case, the convergence time is exactly $p-r+1$ when there are $r > 1$ nearest stored vectors. In particular, $r = 2$ gives convergence time $p-1$.

Fastest convergence by the method in Theorem 22 is achieved when $\epsilon(t) = 1/(a(t)-1)$.

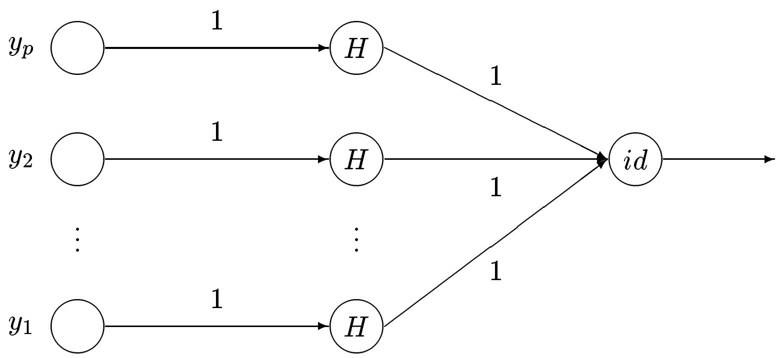

Figure 14: Counting the number of active nodes. The node functions are shown inside the nodes. Here $H(x) = 1$ if $x > 0$, and $H(x) = 0$ otherwise; and $id(x) = x$. The result from the rightmost node is the number of active y-nodes.

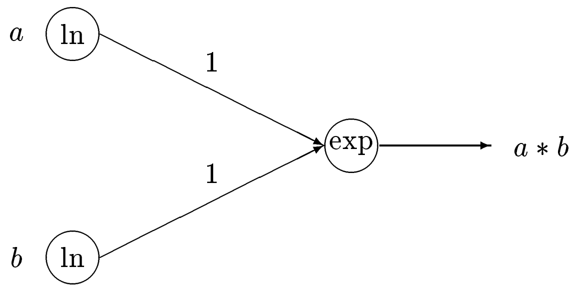

Figure 15: Multiplying a and b when $a, b > 0$. Division a/b is obtained by setting the weight from the b-node to -1.

All this needs is a mechanism that counts the number of active nodes. This can be done neurally (see Figure 14). Also mathematical operations, such as division, can be implemented neurally (see Figure 15).

The proof of Theorem 22 gives inspiration to the following idea: Can we change the node functions so that the activity levels of many more nodes fall below the mean of all the activity levels, i.e., more nodes become inactive in each iteration? We give an example to illustrate this idea.

Example The mean of $y_1 = 0.3$, $y_2 = 0.5$, $y_3 = 0.6$, and $y_4 = 0.9$ is 0.575, but the mean of 0.3^2, 0.5^2, 0.6^2, and 0.9^2 is 0.3775. Thus $y_i - \frac{1}{4}\sum_{i=1}^{4} y_i > 0$ for both y_3 and y_4, but $y_i^2 - \frac{1}{4}\sum_{i=1}^{4} y_i^2 > 0$ for y_4 only. $\square$

Note that $\lim_{s\to\infty} \sqrt[s]{\sum_i x_i^s} = \max\{x_1, x_2, \ldots x_p\}$, so for high powers, all other values than the maximum practically vanish (compared to the maximum value).

The example suggests that raising the activity levels to some higher power could improve the convergence speed. Let us therefore generalize the iteration in formula (27) to

$$y_i(t) = T\left(y_i^s(t-1) - \epsilon(t)\sum_{k\neq i} y_k^s(t-1)\right), \tag{30}$$

or equivalently

$$y_i(t) = T\left(y_i^s(t-1) - \frac{\epsilon(t)}{1+\epsilon(t)}\sum_{k} y_k^s(t-1)\right), \tag{31}$$

where $t \geq 1$ and s is some positive integer. We say that we perform a *power s iteration* (with some specific function ϵ). Formula (27) is thus a power 1 iteration.

We now prove that these modifications are appropriate.

Theorem 23 *Let $a(t)$ be the number of active nodes at the beginning of iteration $t \geq 1$. Assume that $1/(p-t) \leq \epsilon(t) \leq 1/(a(t)-1)$ for $1 \leq t < p$. Then, for each $s \geq 1$, the power s iteration converges in at most $p-1$ iterations to the same solution as the power 1 iteration.*

Proof We get exactly the same dynamics as for formula (30), if the iteration formula is

$$y_i(t) = T^s\left(y_i(t-1) - \epsilon(t)\sum_{k\neq i} y_k(t-1)\right) \tag{32}$$

and the previous network layer raises the initial activity levels to the power s. This means that a power s iteration is equivalent to first making a power 1 iteration and then taking powers of the values resulting from the power 1 iteration. Taking powers of nonnegative values does not change the order of the values (the strictly highest value is still strictly highest, etc.). We can now proceed as in Theorem 22, so the results of that theorem are valid also for power s iterations. $\square$

Again, with $\epsilon(t) = 1/(p-t)$, the convergence time $p-r+1$ is reached when there are $r > 1$ nearest stored vectors. On the other hand, when there is a unique nearest stored vector, a high enough s-value gives immediate convergence. This can be seen as follows. Assume $y_i(0)$ is the highest initial activity level and $y_j(0)$ the second highest. Then $y_j(0) \le y_i(0) - 1/(2n)$. Thus $y_j(1) = 0$ if $y_j^s(0) - \frac{1}{p}\sum_k y_k^s(0) \le 0$. But this is the case if $y_j^s(0) - (y_i^s(0) + y_j^s(0))/p \le 0$, i.e., if $s \ge \ln(p-1)/\ln(1+1/(2ny_j(0)))$. Obviously $y_j(0) \le 1 - 1/(2n)$, so the inequality is satisfied for

$$s \ge \ln(p-1)/\ln(1+1/(2n-1)) \approx (2n-1)\ln(p-1). \tag{33}$$

Next we make a closer examination of the convergence times of power s iterations when there is a unique nearest stored vector. The worst case is the one with $y_i(0) > y_j(0) > 0$ and $y_k(0) = 0$ for all $k \ne i, j$, because the difference $y_i(t) - y_j(t) = y_i^s(t-1) - y_j^s(t-1)$ does not depend on the activity levels of other nodes and having all other nodes zero means that the decrease in activity level in each iteration is as small as possible. The dynamics is fully expressed by the pair of equations

$$\begin{cases} y_i(t) &= y_i^s(t-1) - \epsilon(t) y_j^s(t-1) \\ y_j(t) &= y_j^s(t-1) - \epsilon(t) y_i^s(t-1). \end{cases}$$

Let $\gamma(t) = y_j(t)/y_i(t)$. From the above pair of equations, we get the formula

$$\gamma(t) = \frac{\gamma^s(t-1) - \epsilon(t)}{1 - \epsilon(t)\gamma^s(t-1)}. \tag{34}$$

Convergence is achieved, when $\gamma(t) \le 0$.

Using the recurrence equation (34), we can compute worst-case convergence times for power s iterations in situations with a unique nearest stored vector. A more thorough analysis of the function γ shows that the function is decreasing until convergence is achieved. Hence, the worst case is obtained when $\gamma(0)$ is as large as possible, which corresponds to $y_i(0) = 1$ and $y_j(0) = 1 - 1/(2n)$, i.e., $\gamma(0) = 1 - 1/(2n)$. Note that if only bipolar input vectors are allowed, this maximum value is $1 - 1/n$. The faster $\epsilon(t)$ increases, the faster the convergence. Thus we get an upper bound by taking $\epsilon(t) = 1/(p-t)$.

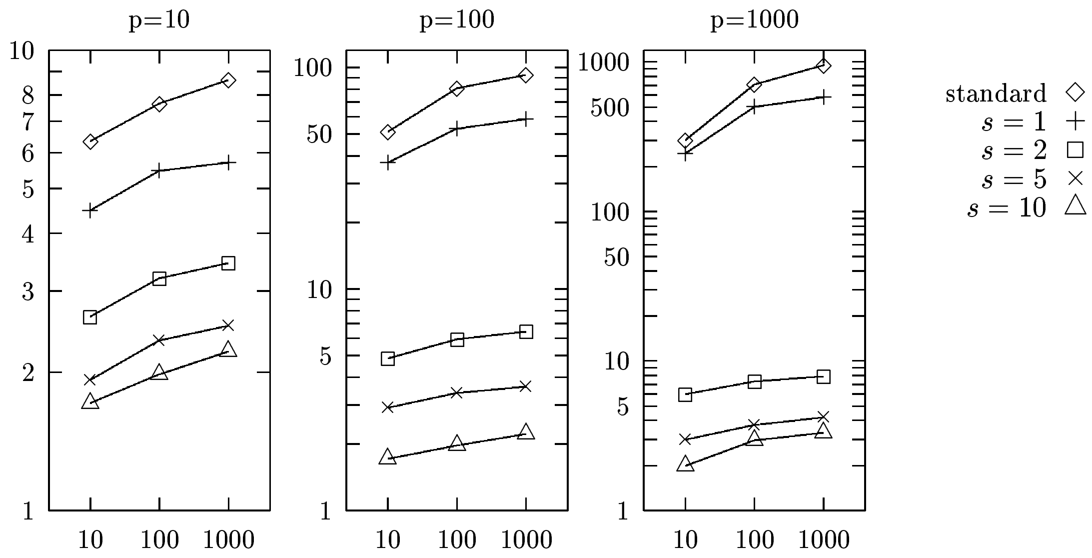

Figure 16: Mean convergence times of the standard Hamming memory with parameter values $\epsilon_i = \frac{1}{p-1}(1 - \frac{1}{n})^{(i-1)/(p-1)}$ and the Hamming memory with power s iterations, s = 1, 2, 5, 10, for p = 10, 100, 1000, and for n = 10, 100, 1000, when the input vectors are bipolar and there is always a unique nearest vector to the input vector. Observe that the axes are logarithmic.

Note that for fixed $\epsilon = 1/(p-1)$, we can solve the recurrence equation for $s = 1$. The result is

$$\gamma(t) = 1 - 2\left(1 + \left(\frac{p-2}{p}\right)^t \left(\frac{2}{1-\gamma(0)} - 1\right)\right)^{-1}.$$

Replacing $\gamma(0)$ by $1 - 1/(2n)$, we get the convergence time

$$\left\lceil \ln(4n-1) \Big/ \ln(1 + 2/(p-2)) \right\rceil . \tag{35}$$

As expected, formula (22) gives the same result.

We made some simulations to demonstrate the convergence times for different WTA networks. The initial activity levels were again produced assuming that the network was given as stored vectors p independent random bipolar vectors of length n and as input vector a random bipolar vector independent of the stored vectors. Consequently, the

Table 10: Convergence times of the standard Hamming memory with $\epsilon_i = \frac{1}{p-1}(1 - \frac{1}{n})^{(i-1)/(p-1)}$ and the Hamming memory with power s iterations, s = 1, 2, 5, and, 10, when the input vectors are bipolar and there is always a unique nearest stored vector. Each entry in the table consists of the mean obtained by the simulations and (in parenthesis) the worst-case bound.

p	n	standard	$s=1$	$s=2$	$s=5$	$s=10$
10	10	6.34 (15)	4.50 (8)	2.64 (4)	1.92 (2)	1.72 (2)
10	100	7.67 (25)	5.47 (9)	3.20 (6)	2.35 (3)	1.98 (3)
10	1000	8.64 (35)	5.71 (9)	3.45 (8)	2.53 (5)	2.22 (4)
100	10	51.4 (163)	37.4 (77)	4.88 (6)	2.92 (3)	2.00 (2)
100	100	81.1 (265)	53.3 (93)	5.96 (9)	3.41 (4)	2.68 (3)
100	1000	93.5 (377)	59.0 (98)	6.43 (12)	3.65 (6)	3.01 (4)
1000	10	303 (1635)	245 (771)	6.00 (7)	3.00 (3)	2.00 (2)
1000	100	713 (2671)	503 (929)	7.29 (10)	3.76 (5)	2.95 (3)
1000	1000	952 (3801)	583 (978)	7.96 (13)	4.22 (6)	3.32 (4)

probability of the proximity between the input vector and a stored vector being i/n is $\binom{n}{i}/2^n$. However, in the simulations, all instances with several nearest stored vectors were excluded.

The results are shown in Figure 16 for p, n = 10, 100, and 1000. The points in the plot are means over 1000 input vectors, except for p = 1000 for the standard Hamming memory and for power 1 iterations, in which case the means are taken over 100 input vectors. The values are also tabulated in Table 10. For comparison, also the worst-case convergence times for the corresponding cases are included in the table. The worst-case values are calculated for bipolar input vectors: For power s iterations they are obtained using the recurrence equation (34) with $\epsilon(t) = 1/(p-t)$ and $\gamma(0) = 1 - 1/n$. The results for the standard Hamming memory are those given in Table 8.

The experiments show that the convergence is very fast for the power s iterations with $s \geq 2$, when there is a unique node with highest initial activity level. In order to avoid situations with several maximum initial activity levels, we could add some (very small) noise to the initial activity levels, and then perform the power s iterations, thus obtaining very fast convergence.

As previously discussed (see formula (33)), immediate convergence would be achieved for $s = \lceil \ln(p-1)/\ln(1+1/(n-1)) \rceil$, but the simulations show that already $s = 2$ gives fast convergence. High powers can give problems with the accuracy: The power s iterations with s big give very small activity levels, which can be problematic as the accuracy is limited in real situations. Consequently, low powers can be recommended.

6 A New Memory Model

We can extend the power s iterations in Section 5.4 to other functions than powers as well. And we already saw that high powers give essentially immediate result. Inspired by these results, we are ready to take the final step: to get rid of the iterations altogether. The resulting model we describe here, simply called the "new memory model," is a feed-forward network that produces ternary output vectors.

The model is described in Section 6.1, a collection of different node functions is given in Section 6.2, and some simulations are reported in Section 6.3.

6.1 Architecture

Recall that in Section 2.3 we proposed a linear combination of possible vectors as a result in unclear cases. Our new memory model produces its result according to such a combination, with emphasis on vectors with high proximity values.

As before, we denote the stored vector pairs by (u^1, v^1), (u^2, v^2), $\ldots$, (u^m, v^m), the input vector by x, and the output vector by z. The stored vectors are bipolar; the input and output vectors are ternary. We assume that all vectors u^i are different from one another. The network works as an autoassociative memory when $v^i = u^i$ for all i.

The new network, shown in Figure 17, is like the standard Hamming network in Figure 8, but with only one layer of intermediate nodes. We get rid of the WTA component altogether. There is a connection with weight $a_{ij} = u^i_j/(2n)$ from each input node j to each intermediate node i, and there is a connection with weight $b_{ij} = v^j_i$ from each intermediate node i to each output node j. If the number of stored vector pairs m is less than the capacity p, the weights are chosen as if the rest of the stored vectors pairs were $(\overline{0}, \overline{0})$. The distance measuring nodes have threshold values $-1/2$ and the identity function as node function, the intermediate nodes have threshold values 0 and node function f, and the output nodes have threshold values 0 and node function S_η, where the tolerance parameter η can be any positive real-valued number. As this is a feed-forward network, there is no convergence property to examine.

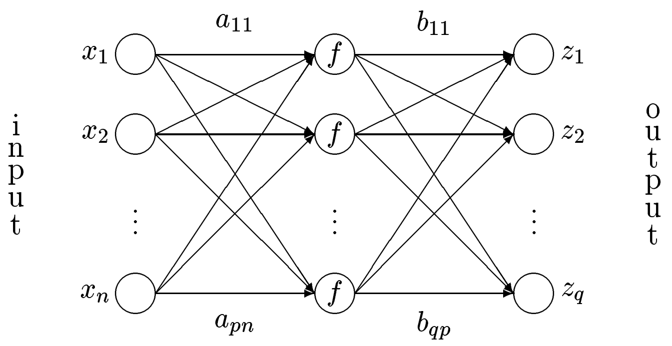

Figure 17: The new network model. Compare with Figure 8.

Our new memory model satisfies the following requirements:

1. If the input vector is identical to some stored vector, the result should be determined by this stored vector alone; i.e., if $x = u^i$ for some i, then $z = v^i$.

2. If there are several stored vectors that are near the input vector, the result should be based on a linear combination; i.e., if there is a set $\{ i_1, i_2, \ldots, i_r \}$ such that $d(x, u^{i_1}) \approx d(x, u^{i_2}) \approx \cdots \approx d(x, u^{i_r}) \ll d(x, u^j)$ for all $j \notin \{ i_1, i_2, \ldots, i_r \}$, then z is constructed from v^{i_1}, v^{i_2}, $\ldots$, v^{i_r}, possibly with some elements set to zero.

3. If all stored vectors are far from the input vector, the result should be $(0, 0, \ldots, 0)$; i.e., if $d(x, u^i)$ is large for all i, then $z = \overline{0}$.

The output of the network is

$$z = S_\eta \left(\sum_{i=1}^{p} f(\frac{x^{\mathrm{T}} u^i + n}{2n}) v^i \right), \tag{36}$$

i.e., a linear combination of the stored output vectors is cut off into values -1, 0, and 1.

The requirements listed above are realized by choosing appropriate weight functions f. The weight functions f are strictly increasing over the set of possible argument values (proximity values) and are here chosen so that $f(0) = 0$ and $f(1) = 1$. The requirement $f(0) = 0$ is not essential. In fact, we could even allow small negative values for $f(0)$. If s is small, say $s \ll 0.5$, then $f(s)$ is so small that the corresponding stored vector cannot influence the result. Consequently, the small negative values have no practical importance, and we restrict our study to functions with $f(0) = 0$.

We now turn to the first requirement on the list above (if $x = u^i$ for some i, then $z = v^i$). Let us study z_j for some arbitrary element j. Now if $x = u^i$ for some i, then vector u^i contributes to element j of the sum in formula (36) by the amount v_j^i. In the worst possible case $m = p$ and $v_j^k = -v_j^i$ for all $k \neq i$. We assumed that the u-vectors are different, so the vectors u^k are different from x for all $k \neq i$. The highest possible proximity value less than 1 is $1 - 1/n$, because there are no zero valued elements in x (as $x = u^i$). The $p - 1$ vectors u^k, where $k \neq i$, can thus contribute to element j of the sum in formula (36) by at most the amount $(p-1)f(1-1/n)(-v_j^i)$, and from the requirement $v_j^i = S_\eta(f(1)v_j^i - (p-1)f(1-1/n)v_j^i)$ we get the inequality $f(1) - (p-1)f(1-1/n) > \eta$, which is equivalent to

$$f\,(1 - 1/n) < \frac{1-\eta}{p-1}. \tag{37}$$

The other requirements are less stringent.

6.2 Some Possible Weight Functions

In the following we examine some possible weight functions. We give some motivation for them and study their properties. The functions we study are of the following forms:

1. function form $f(s) = (s+\alpha)^k$, where α, $k > 0$,
2. function form $f(s) = e^{ks}$, where $k > 0$,
3. function form $f(s) = \ln(1-s)$,
4. function form $f(s) = (1-s)^{-k}$, where $k > 0$.

These function types are of course not the only possible ones, but here it is possible to choose only a few promising functions for further study. Combinations of these functions are naturally also possible as weight functions.

1. The first function form is $f(s) = (s+\alpha)^k$. The normalization criteria $f(0) = 0$ and $f(1) = 1$ give us the function

   $$f(s) = \frac{(s+\alpha)^k - \alpha^k}{(1+\alpha)^k - \alpha^k}. \tag{38}$$

   Even for modest p and n, formula (37) requires very big exponents k. Relation (37) is satisfied when $(1 - 1/((\alpha+1)n))^k < (1-\eta)/(p-1)$, i.e., when

   $$k > \ln\left(\frac{1-\eta}{p-1}\right) \Big/ \ln\left(1 - \frac{1}{(\alpha+1)n}\right) \approx (\alpha+1)n\ln(p-1). \tag{39}$$

   With this weight function and $\alpha = 0$ the network behaves essentially like a standard Hamming network, yet so that zero valued elements are possible in the output, and if there are several stored vectors at minimum distance from the input vector, then all those nearest stored vectors are taken into account when computing the result. To see that it behaves almost like a standard Hamming memory, take $\eta = 0$ and assume there is a unique nearest stored vector $\tilde{u}$ to the input vector x (the pair $(\tilde{u}, \tilde{v})$ is stored), and let $\tilde{d} = (x^{\mathrm{T}}\tilde{u} + n)/(2n)$. Now by formula (37) we get

   $$\begin{aligned} f(\tilde{d} - 1/n) &= (\tilde{d} - 1/n)^k \leq \tilde{d}^k(1-1/n)^k \\ &= f(\tilde{d})f(1-1/n) < \frac{f(\tilde{d})}{p-1}. \end{aligned} \tag{40}$$

Thus $z = \tilde{v}$.

2. The second function form is $f(s) = e^{ks}$. After normalization we get the function

$$f(s) = \frac{e^{ks} - 1}{e^k - 1}. \tag{41}$$

This weight function behaves essentially like the previous one. For big k we have $f(s) \approx e^{k(s-1)}$, so formula (37) gives k approximately greater than $n \ln(p-1)$. If we disregard the requirement $f(0) = 0$, we could use directly the function $f(s) = e^{k(s-1)}$.

The function form (41) also results in a behavior almost like that of the standard Hamming memory. To see this, take $\eta = 0$ and assume there is a unique nearest stored vector $\tilde{u}$ (the pair $(\tilde{u}, \tilde{v})$ is stored), and let $\tilde{d} = (x^{\mathrm{T}}\tilde{u} + n)/(2n)$. Now by formula (37) we get

$$\begin{aligned} f(\tilde{d} - 1/n) &= \frac{e^{k(\tilde{d}-1/n)} - 1}{e^k - 1} \leq \frac{e^{k\tilde{d}} - 1}{e^k - 1} \frac{e^{k(1-1/n)} - 1}{e^k - 1} \\ &= f(\tilde{d}) f(1 - 1/n) < \frac{f(\tilde{d})}{p - 1}. \end{aligned} \tag{42}$$

Thus $z = \tilde{v}$.

3. Suppose we instead of the functions of type $f(s) = (s + \alpha)^k$ try to use functions of type $f(s) = 1 - \sqrt[k]{1 - s}$. As

$$\begin{aligned} (1 - s)^{\frac{1}{k}} &= 1 - \frac{1}{k}s - \frac{k-1}{2k^2}s^2 - \frac{2k^2 - 3k + 1}{6k^3}s^3 - \dots \\ &\approx 1 - \frac{s}{k} - \frac{s^2}{2k} - \frac{s^3}{3k} - \dots, \end{aligned}$$

we see that

$$f(s) \approx \frac{1}{k}(s + \frac{s^2}{2} + \frac{s^3}{3} + \dots) = -\frac{1}{k} \ln(1 - s). \tag{43}$$

The next function form we study is therefore

$$f(s) = \begin{cases} -\alpha \ln(1 - s), & \text{if } s < 1; \\ 1, & \text{if } s = 1. \end{cases} \tag{44}$$

From formula (37) we get the requirement $\alpha < (1 - \eta)/((p - 1) \ln n)$. However, this function does not grow fast enough; for instance, $f(0.9)/f(0.8) \approx 1.43$. We can compensate this by modifying the function into

$$f(s) = \begin{cases} \alpha(-\ln(1-s))^k, & \text{if } s < 1; \\ 1, & \text{if } s = 1. \end{cases} \tag{45}$$

Now formula (37) gives us the requirement

$$\alpha < \frac{1-\eta}{(p-1)\ln^k n}. \tag{46}$$

If we, for instance, would like to have $f(0.9)/f(0.8) = p/2$, we take

$$k = \ln(p/2) \Big/ \ln\left(\frac{\ln 0.1}{\ln 0.2}\right) \approx 2.792 \ln(p/2). \tag{47}$$

However, in order to have formula (45) strictly increasing over all possible proximity values, also the inequality $\alpha \ln^k(2n) < 1$ must be satisfied.

4. Our last function form is $f(s) = (1-s)^{-k}$. The normalization gives

$$f(s) = \alpha\left(\frac{1+\alpha}{(1-s)^k+\alpha} - 1\right) = \frac{1-(1-s)^k}{1+\alpha^{-1}(1-s)^k}, \tag{48}$$

where by formula (37),

$$\alpha^{-1} > n^k\left(\frac{p-1}{1-\eta}(1-1/n^k) - 1\right). \tag{49}$$

An example of a function of this type is shown in Figure 18. For big p and n, the parameter α is very small, so $f(s) \approx \alpha((1-s)^{-k} - 1)$, when $s \neq 1$. From the approximation we also see that for a given β the ratio $f(1-\beta s)/f(1-s)$ is almost constant for all possible values of s, for example, $f(0.95)/f(0.9) \approx f(0.9)/f(0.8) \approx f(0.8)/f(0.6)$. Now we can determine k according to how much we want individual function values to influence the result. If we would like to have $f(0.9)/f(0.8) = p/2$, meaning that we need $p/2$ values $f(0.8)$ to overrule a value $f(0.9)$, we arrive at the approximate requirement $(10^k-1)/(5^k-1) = p/2$, which is approximately equivalent to $2^k = p/2$, i.e.,

$$k \approx \log_2 p - 1. \tag{50}$$

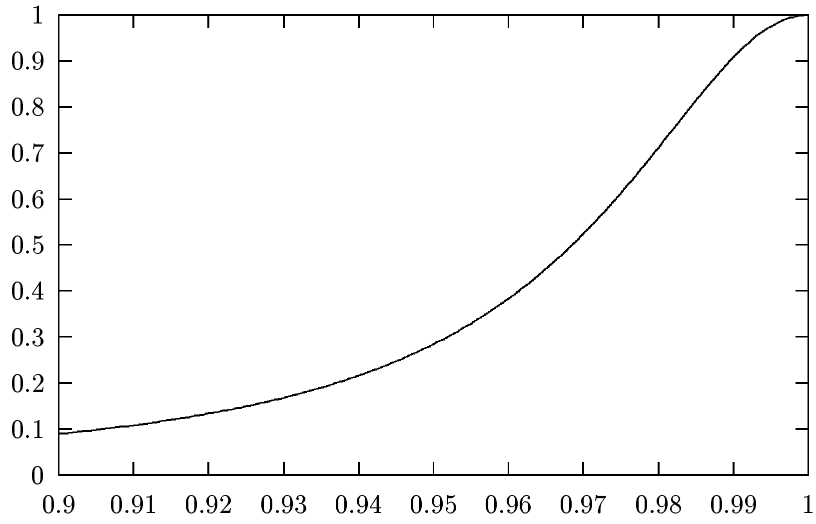

Figure 18: A plot of part of the function $f(s) = \frac{1-(1-s)^2}{1+1000(1-s)^2}$.

In Figure 19 we illustrate the differences between the function forms. The functions in the figure are $f^{(1)}(s) = s^{21}$, $f^{(2)}(s) = (e^{22s}-1)/(e^{22}-1)$, $f^{(3)}(s) = 0.00262\cdot(-\ln(1-s))^{4.49}$, and $f^{(4)}(s) = (1-(1-s)^{2.32})/(1+(1-s)^{2.32}/5.98\cdot 10^{-4})$. These functions would result if we use the choices on parameter values sketched above with $p = n = 10$ and $\eta = \frac{\sqrt{p}}{2} f(5/8)$. The value of η is here chosen according to the same formula as in the simulation experiments in the next section.

The high parameter values above were caused by the requirement (37). A way to get smaller parameter values is to preprocess the proximity values, which are given as input to f. If we use the angle between x and u as a measure of the closeness between x and u, we get weight functions $g(s) = f(\frac{2}{\pi}\arcsin s)$.

In common for all our function forms are the small resulting $f(s)$-values, so how can we take care of the accuracy? One solution is to represent the activity levels by their logarithms. When computing the node function, the logarithm of it can be taken; when this logarithm of the actual activity level is received by a node, the exponential of it can be computed. For some node functions this is easy to do: If $f(s) = \alpha(-\ln(1-s))^k$, then $\ln f(s) = \ln\alpha + k\ln(-\ln(1-s))$. However, the logarithms must be transformed to the

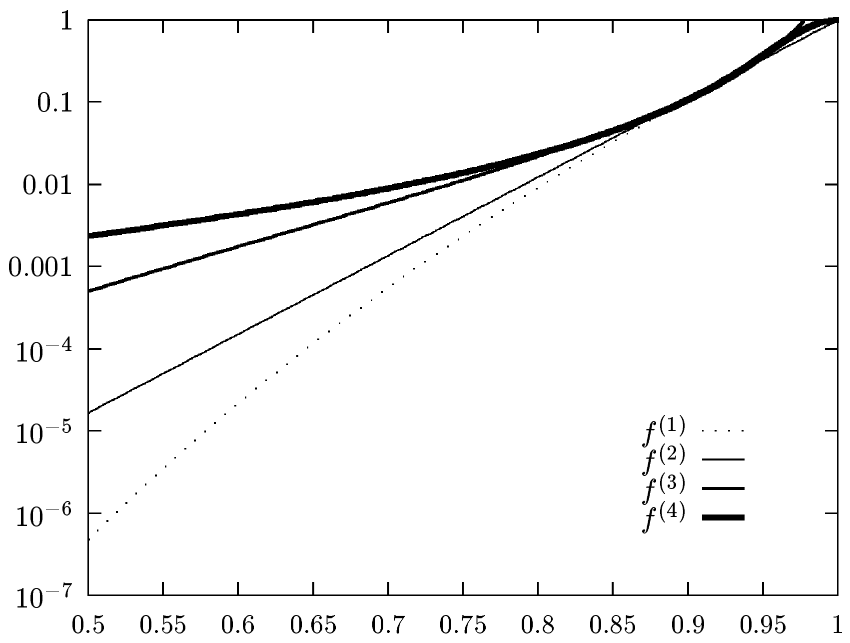

Figure 19: Different weight functions. The expressions for the functions can be found in the text. Note that the scale on the ordinate axis is logarithmic.

actual values for the summation performed in the nodes. The summation is especially sensitive to errors due to limited accuracy.

Note that function forms 1 and 2 give rise to a behavior almost that of the standard Hamming memory, and thus this model can often be used instead of the standard model. An advantage with the new memory over the standard Hamming network is that it works in constant time (assuming that the node functions are computable in unit time). The node functions are, however, involved. This does not mean that they cannot be used in practice. The main problem in (electrical) hardware implementations is not the complexity of the node functions, but that they are in fact only roughly approximated, which can give rise to problems due to approximation errors.

6.3 Simulations for the New Memory Model

In order to test different weight functions in the new memory model, 40 vectors of length 35 were stored. The network was used as an autoassociative memory. Our 40 vectors are shown in Appendix III. The vectors were taken from the display of a calculator and consist of the ten digits, all the 26 alphabetic characters, and the symbols "+", "−", "*", and "/". In the following, this collection of vectors is called the *Calculator Test Set.* The stored vectors are highly correlated, the mean mutual distance (in 1-metric) is 29, the smallest distance is 4, and the largest distance is 52.

The input vector sets were generated by corrupting the stored vectors. For each stored vector, 25 replicates were made. For each of these, a certain number of randomly chosen elements were changed, either to their opposite values or to zero. The input vector sets with elements changed to zero were generated with the purpose of illustrating the benefits of including "don't know" elements in the input vectors. This way six input vector sets each of $25 \times 40 = 1000$ vectors were obtained:

- 3 elements changed to their opposite values,
- 6 elements changed to zero ,
- 6 elements changed to their opposite values,
- 12 elements changed to zero,
- 9 elements changed to their opposite values,
- 18 elements changed to zero,

Moreover, for the two sets of 6 changed elements, exactly the same elements were changed, however to different values. Note that 18 is more than half of the vector length.

The input vector sets were fed into a (simulated) new memory model with different weight functions, and each resulting vector was compared with the respective stored vector from which the input vector was originally generated.

The weight functions $f(s)$ and the values of η used in this experiment were the following:

- Pow denoting $f(s) = s^{127}$ and $\eta = 3.77 \cdot 10^{-26}$;
- Log denoting $f(1)=1$, $f(s)=6.36{\cdot}10^{-7}(-\ln(1-s))^{8.36}$, for $s<1$, and $\eta=1.71\cdot10^{-6}$;
- Frac denoting $f(s) = \frac{1-(1-s)^{4.32}}{1+\alpha^{-1}(1-s)^{4.32}}$, with $\alpha = 5.62 \cdot 10^{-9}$, and $\eta = 1.21 \cdot 10^{-6}$.

The parameters were chosen according to the formulae in Section 6.2, but tolerance η was chosen to be $\frac{\sqrt{p}}{2}f(5/8)$. The choice of value of parameter η depends much on what kind of behavior we desire. Higher values give more zero valued elements. Different promising formulae for η were tested and the one used here proved to work satisfactorily. The factor $\sqrt{p}$ is natural, because the standard deviation of p equally distributed independent random variables is $\sqrt{p}$ times the standard deviation for one of the variables.

The exponential weight function (function form number 2) was not tested, because it works almost like the power weight function (Pow).

Table 11 shows the results. When all three weight functions gave rise to the same results, they were gathered under the label All. Note that we used the 1-metric for computing the vector differences. The minimum distances are not shown in the table, because they were zero in all cases.

By comparing the results for input vectors with elements changed to zero with the results for input vectors with elements changed to their opposite values, we see that using zero valued elements in the input vectors is very effective, as expected. Study especially vectors with equal distances to the stored vectors, but one with elements changed to opposite values and the other with elements changed to zeros. For instance, compare the columns for Pow with 9 elements changed to opposite values with Pow with 18 elements changed to zero. As noted in the preceding section, Pow works in practice like the standard Hamming memory when tolerance η is small. In fact, the resulting vectors for Pow did not contain any zeros. This means that we, in effect, can interpret the results for Pow as results for the standard Hamming memory. We also see from the table that the results when using other weight functions do not differ much from the results for Pow when only a few elements are corrupted. The more corrupted elements, the bigger the differences.

Table 11: Differences between stored and resulting vectors in the Calculator Test.

Function	POW	LOG	FRAC	ALL	POW	LOG	FRAC	POW	LOG	FRAC
Bits changed	3	3	3	6	6	6	6	12	12	12
Changed to	opp.	opp.	opp.	zero	opp.	opp.	opp.	zero	zero	zero
Mean	0.08	0.08	0.08	0.01	0.85	1.12	1.72	0.09	0.21	0.72
Max	10	10	10	6	22	22	20	8	8	8

Function	POW	LOG	FRAC	POW	LOG	FRAC
Bits changed	9	9	9	18	18	18
Changed to	opp.	opp.	opp.	zero	zero	zero
Mean	4.54	5.44	9.32	0.29	1.48	6.85
Maximum	30	30	28	16	16	20

A first glance at the results seems to indicate that POW works best of these three weight functions. Nevertheless, the weight functions LOG and FRAC are useful and have desired features. In Table 12, we can see for three input vectors the corresponding results when different weight functions and different values of tolerance η are applied. The different η-values are chosen to illustrate how sensitive the results are to the values of η. In the figure, ■ means value $+1$, - means value -1, and 0 means value 0. The input vectors are the vector consisting of only values $+1$, the vector consisting of only values -1, and a corrupted version of "E" with nine elements changed to their opposite values. The experiment was conducted also for corrupted versions of "E" with three elements changed to opposite and with six elements changed to zero, but these experiments are not shown in the table, because in all nine cases of different weight functions and values of η, we got perfect letters "E" as output.

Table 12: Some difficult input and the corresponding output vectors for different weight functions and different values of η.

Function	POW	LOG	FRAC	POW	LOG	FRAC	POW	LOG	FRAC
Value of η	0	0	0	$3.77 \cdot 10^{-26}$	$1.71 \cdot 10^{-6}$	$1.21 \cdot 10^{-6}$	$1.00 \cdot 10^{-9}$	$1.00 \cdot 10^{-5}$	$4.00 \cdot 10^{-6}$
Input	Output vectors			Output vectors			Output vectors		
■■■■■ ■■■■■ ■■■■■ ■■■■■ ■■■■■ ■■■■■ ■■■■■	■■■■- ■---■ ■---■ ■■■■- ■---■ ■---■ ■■■■-	■■■■- ■---■ ■---■ ■■■■■ ■---■ ■---■ ■■■■-	■■■■- ■---■ ■---■ ■-■■■ ■---■ ■---■ ■■■■-	00000 00000 00000 00000 00000 00000 00000	00000 00000 00000 00000 00000 00000 00000	00000 ■---■ ■---0 ■0■00 ■---0 ■---0 00000	00000 00000 00000 00000 00000 00000 00000	00000 00000 00000 00000 00000 00000 00000	00000 00000 00000 00000 00000 00000 00000
----- ----- ----- ----- ----- ----- -----	----- ----- ----- ■■■■■ ----- ----- -----	----- ----- ----- --■-- ----- ----- -----	----- ----- ----- --■-- ----- ----- -----	----- ----- ----- ■■■■■ ----- ----- -----	----- ----- ----- --■-- ----- ----- -----	----- ----- ----- --■-- ----- ----- -----	----- ----0 ---0- 00■00 -0--- 0---- -----	----- ----- ----- 0-■0- ----- ----- -----	----- ----0 ----- 0-■0- ----- 0---- -----
■■-■■ ■---- ■-■■■ ■■■■- ■-■■■ ----■ ■■■■■	■■■■- ■---■ ■---■ ■■■■- ■---■ ■---■ ■■■■■	■■■■- ■---■ ■---■ ■■■■- ■---■ ■---■ ■■■■■	■■■■- ■---■ ■---■ ■■■■- ■---■ ■---■ ■■■■-	■■■■- ■---■ ■---■ ■■■■- ■---■ ■---■ ■■■■■	■■■■0 ■---■ ■---■ ■■■■- ■---■ ■---■ ■■■■0	■■■■- ■---■ ■---0 ■■■■- ■---■ ■---■ ■■■■0	00000 00000 00000 00000 00000 00000 00000	■■■■0 ■---0 ■---0 ■■■■- ■---0 ■---0 ■■■■0	■■■■0 ■---0 ■---0 ■0■■- ■---0 ■---0 0■■00

First examine the three cases with tolerance $\eta = 0$. Study the results for the first input vector. The stored vector with most values +1, namely 20, is "B", which we get using function POW. There are, however, several stored vectors with 18 or 19 elements with values +1, and as the node functions LOG and FRAC are not as steep as POW, these other stored vectors influence the results of LOG and FRAC, so that they do not give perfect

```
■■■■■        00000        ■■■■-        ■■■■-
■■■■■        ■---■        ■---■        ■---■
■■■■■        ■---0        ■---■        ■---■
■■■■■   ⟹    ■0■00   ⟹    ■■■■0   ⟹    ■■■■-
■■■■■        ■---0        ■---■        ■---■
■■■■■        ■---0        ■---■        ■---■
■■■■■        00000        ■■■■-        ■■■■-
```

Figure 20: Example of the iterative use of the new memory model with Frac and $\eta = 1.21 \cdot 10^{-6}$.

letters "B". Now consider the results for the second input vector. The stored vectors with least elements with values $+1$, namely 5, are "$-$" and "/". Note that "$-$" and "/" have one element in common, and this is exactly the result for Log and Frac. Now study the results for the third input vector. The "E" is corrupted in such a way that it is equally far from "B" than from "E". Moreover, "B" and "E" are very similar. We get almost vectors "B" as result.

Consider the three cases with tolerance $\eta = \frac{\sqrt{p}}{2} f(5/8)$. This is the formula used in the experiment in Table 11. Study the results for the first input vector. As we already observed, there are no stored vectors near this input vector. These values of η are for Pow and Log big enough to give a complete "don't know" result. For Frac the value of η is still so small that we get a fragment of "B". Study the results for the other two input vectors. Here we have very small changes compared to $\eta = 0$.

Consider now the case with high values on the tolerance parameter η. The first input vector gives in all cases a complete "don't know" result. Study the results for the second input vector. As already noted, both "$-$" and "/" are equally near the input vector, with the element in the middle in common. This is reflected in the results. Especially the result for Pow is excellent. Finally look at the third input vector. Here we get a "don't know" result for Pow. This is a proper result, as the input vector is equally far from "B" and "E". However, "B" and "E" have much in common, so the results for Log and Frac are even better. Here zero valued elements fill the parts where "B" and "E" disagree.

This experiment shows that the new memory model produces appropriate results. It also shows that the choice of value for parameter η is crucial.

The memory could also be used iteratively: By repeatedly giving the result as input, we can come closer and closer to the optimal result (as given by the standard Hamming memory). For instance, with FRAC and tolerance $\eta = 1.21 \cdot 10^{-6}$, we get the series of vectors shown in Figure 20.

7 Discussion

We have studied computational complexity questions for the Hopfield and Hamming neural associative memories.

One theme in this thesis was the extension from bipolar vectors to ternary vectors with "don't know" elements. This extension is rather straightforward in the input, but in the output we had to replace the signum function with a function also giving zero as result. This new function S_η works like the signum function, except that it gives zero as result when the absolute value of its argument is at most η.

For Hopfield memories, "don't know" elements in the input proved to be useful, but they are of less practical importance in the output.

We proved bounds on the convergence times in Hopfield networks for both binary and ternary vectors in both asynchronous and synchronous updating. In asynchronous updating, the result is always a stable vector (if the connection matrix has nonnegative diagonal elements). In synchronous updating, the result can also be a cycle of length two. Our bound on the convergence time for bipolar vectors in asynchronous updating is about half the previously known value. For ternary vectors, the bound is about four times as large as for bipolar vectors. The bounds on the convergence time are about the same for synchronous updating as for asynchronous updating. However, the bounds we obtained are not tight in general, and a challenging task would be to improve them substantially. Also average-case convergence times would be interesting to obtain, but this is likely to be much more difficult. Simulations with asynchronous updating showed that when only a few vectors are stored, the Hopfield memory works reliably and converges fast.

Our main theme in the study of Hopfield memories was the computational complexity of assessing the quality of a given network. Our quality criteria were: the number of spurious stable vectors, the size of the attraction domain of a given stable vector (in synchronous updating), and the attraction radius of a given stable vector (in synchronous updating). We distinguished between three types of Hopfield networks based on the (non)existence of zero valued elements in the input and the output: bipolar (with bipolar input and output), ternary standard (with ternary input and bipolar output), and ternary (with ternary input and output). All our quality criteria turned out to be NP-hard to evaluate for all three categories of networks. In fact, the problems of counting the stable vectors and the sizes of attraction domains are $\#P$-complete, and even approximating the attraction radius is hard (unless $P = NP$). Hence, unless $P = NP$, there are no exact polynomial time computable formulae for these quantities.

The NP-completeness results are valid for connection matrices produced by the tensor product storage rule with each vector allowed to be stored exponentially many times; an open problem is whether this holds also for matrices produced by the tensor product rule with each vector stored only once or even polynomially many times.

The limited capacity of the Hopfield memory diminishes the interest in this model. However, the memory's intriguing dynamics and position as a form of basic associative memory structure justifies the study of Hopfield networks.

The main part of the Hamming network is the winner-take-all component, in which the computation is iterated until at most one node remains active. We established which criteria the parameters of the network must satisfy in order to guarantee convergence, and we obtained a tight bound on the convergence time. This is our main result about Hamming memories. We also derived which conditions the parameters must satisfy in order to minimize the worst-case convergence time. With the optimal parameter values, the worst-case convergence time is $\Theta(p \log(pn))$, where p is the memory capacity and n the vector length. Restricting the instances to those with a unique nearest stored vector to the input vector, the worst-case convergence time is $\Theta(p \log n)$. The average-case convergence time for the Hamming network is an open problem; simulation results indicate that it is about $O(p)$.

We also showed, for random vectors, that if the number of vectors is subexponential in n, the probability that there is only one nearest stored vector approaches 1 as n grows. This, however, does not mean that the situations with several nearest stored vectors can be ignored.

By changing the connection weights during the computation, we were able to speed up the convergence considerably simultaneously introducing "don't know" results in the output. The worst-case convergence time drops to $p-1$. Then we defined a new concept: power s iterations. Already power 2 iterations gave drastically faster convergence. We provided also a method for computing the worst-case convergence times for power s iterations when there is a unique nearest stored vector; when there are several nearest stored vectors, the worst-case convergence time is $p-1$. An average-case analysis for the power s iterations are not especially interesting, as already the upper bounds give very low values.

The results for Hamming networks were formulated for ternary input vectors, but the results are valid also for binary vectors (it is sufficient to change $2n$ to n in the formulae).

Finally, we introduced a new memory model based on the Hamming memory, but with a feed-forward network structure made possible by the use of more complicated node functions. The new memory model gives ternary vectors as result. Simulation experiments showed that the model behaves as expected. More work should be done on the effects of different node functions, and also on adjusting the parameter η.

In this thesis, we adopted a purely theoretical point of view — the computations are performed with unlimited accuracy. In practice, limited accuracy and signal errors are important; this is an area we left unexplored.

References

[1] Y. S. Abu-Mostafa and J.-M. St.Jacques, "Information capacity of the Hopfield model," *IEEE Transactions on Information Theory* 31 (1985), 461–464.

[2] S.-I. Amari, "Characteristics of randomly connected threshold-element networks and network systems," *Proceedings of the IEEE* 59 (1971), 35–47.

[3] S.-I. Amari, "Learning patterns and pattern sequences by self-organizing nets of threshold elements," *IEEE Transactions on Computers* 21 (1972), 1197–1206.

[4] S.-I. Amari, "A method of statistical neurodynamics," *Kybernetik* 14 (1974), 201–215.

[5] S.-I. Amari, "Neural theory of association and concept-formation," *Biological Cybernetics* 26 (1977), 175–185.

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

[7] S.-I. Amari and K. Maginu, "Statistical neurodynamics of associative memory," *Neural Networks* 1 (1988), 63–73.

[8] D. J. Amit, *Modeling Brain Function: The World of Attractor Neural Networks.* Cambridge University Press, Cambridge, Great Britain, 1989.

[9] 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 (1985), 1530–1533.

[10] J. A. Anderson, "A memory storage model utilizing spatial correlation functions," *Kybernetik* 5 (1968), 113–119.

[11] J. A. Anderson, "A simple neural network generating an interactive memory," *Mathematical Biosciences* 14 (1972), 197–220.

[12] P. Baldi, "Neural networks, acyclic orientations of the hypercube, and sets of orthogonal vectors," *SIAM Journal on Discrete Mathematics* 1 (1988), 1–13.

[13] P. Baldi, "Neural networks, orientations of the hypercube, and algebraic threshold functions," *IEEE Transactions on Information Theory* 34 (1988), 523–530.

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

[15] E. K. Blum, "Mathematical aspects of outer-product asynchronous content-addressable memories," *Biological Cybernetics* 62 (1990), 337–348.

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

[17] J. Bruck and J. W. Goodman, "On the power of neural networks for solving hard problems," *Journal of Complexity* 6 (1990), 129–135.

[18] J. Bruck and V. P. Roychowdhury, "On the number of spurious memories in the Hopfield model," *IEEE Transactions on Information Theory* 36 (1990), 393–397.

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

[20] S. Cajal and Y. Ramon, *Histology du Système Nerveux.* CSIC, Madrid, Spain, 1908; reprinted 1972.

[21] A. C. C. Coolen and H. J. J. Jonker, "Size of the domains of attraction in the Hopfield model," *Physical Review A* 40 (1989), 5295–5298.

[22] M. Cottrell, "Stability and attractivity in associative memory networks," *Biological Cybernetics* 58 (1988), 129–139.

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

[24] J. D. Cowan and D. H. Sharp, "Neural nets," *Quarterly Reviews of Biophysics* 21 (1988), 365–427.

[25] DARPA, *Neural Network Study.* AFCEA International Press, Fairfax, Virginia, 1988.

[26] S. DasGupta, A. Ghosh, and R. Cuykendall, "Convergence in neural memories," *IEEE Transactions on Information Theory* 35 (1989), 1069–1072.

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

[28] E. Domany and H. Orland, "A maximum overlap neural network for pattern recognition," *Physics Letters A* 125 (1987), 32–34.

[29] N. H. Farhat, D. Psaltis, A. Prata, and E. Paek, "Optical implementation of the Hopfield model," *Applied Optics* 24 (1985), 1469–1475.

[30] M. Fleisher, "The Hopfield model with multi-level neurons," in *Proc. Neural Information Processing Systems (1987)* ed. D. Z. Anderson. American Institute of Physics, New York, 1988, 278–289.

[31] P. Floréen, "An analysis of the convergence time of Hamming memory networks," in *Proc. International Joint Conference on Neural Networks 1990.* IEEE, New York, 1990, Vol. I, 867–872.

[32] P. Floréen, *Computational Complexity Issues in Neural Associative Memories.* Licentiate Thesis. University of Helsinki, Department of Computer Science, Report C–1990–40. ISBN 951–45–5562–7. 73+3 pages.

[33] P. Floréen, "The convergence of Hamming memory networks," *IEEE Transactions on Neural Networks* 2 (1991), 449–457.

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

[35] P. Floréen, "A new neural associative memory model," *International Journal of Intelligent Systems* 7 (1992), 455–467.

[36] P. Floréen and T. Huuskonen, "Uniqueness of maximum values in discrete distributions," manuscript, 1992.

[37] P. Floréen and P. Orponen, "Counting stable states and sizes of attraction domains in Hopfield nets is hard," in *Proc. International Joint Conference on Neural Networks 1989.* IEEE, New York, 1989, Vol. I, 395–399.

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

[39] P. Floréen and P. Orponen, "Attraction radii in Hopfield nets are hard to compute," submitted for publication, 1992.

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

[41] F. Fogelman Soulie and G. Weisbuch, "Random iterations of threshold networks and associative memory," *SIAM Journal on Computing* 16 (1987), 203–220.

[42] E. Gardner, "Maximum storage capacity in neural networks," *Europhysics Letters* 4 (1987), 481–485.

[43] E. Gardner, "The space of interactions in neural network models," *Journal of Physics A* 21 (1988), 257–270.

[44] E. Gardner and B. Derrida, "Optimal storage properties of neural network models," *Journal of Physics A* 21 (1988), 271–284.

[45] 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.

[46] 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, University of Toronto, Department of Computer Science, 1988.

[47] E. Goles, "Fixed point behavior of threshold functions on a finite set," *SIAM Journal on Algebraic Discrete Methods* 3 (1982), 529–531.

[48] E. Goles Ch., "Dynamics of positive automata networks," *Theoretical Computer Science* 41 (1985), 19–32.

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

[50] 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.

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

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

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

[54] E. Goles and G. Y. Vichniac, "Lyapunov functions for parallel neural networks," in *Proc. Neural Networks for Computing,* ed. J. Denker. AIP Conference Proceedings Vol. 151, American Institute of Physics, New York, 1986, 165–181.

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

[56] M. Hall Jr., *Combinatorial Theory.* Wiley, New York, first edition, 1967; second edition, 1986.

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

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

[59] M. H. Hassoun and P. B. Watta, "Exact associative neural memory dynamics utilizing Boolean matrices," *IEEE Transactions on Neural Networks* 2 (1991), 437–448.

[60] D. O. Hebb, *The Organization of Behavior.* Wiley, New York, 1949.

[61] R. Hecht-Nielsen, *Neurocomputing.* Addison-Wesley, Reading, Massachusetts, 1990.

[62] 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, California, 1991.

[63] M. Hofri, *Probabilistic analysis of algorithms.* Springer-Verlag, New York, 1987.

[64] 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.

[65] 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 (1984), 3088–3092.

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

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

[68] G. M. Jacyna and E. R. Malaret, "Classification performance of a Hopfield neural network based on a Hebbian-like learning rule," *IEEE Transactions on Information Theory* 35 (1989), 263–280.

[69] "Japan moves toward the information society," *IEEE Expert* April 1992, 54–58.

[70] A. Johannet, L. Personnaz, G. Dreyfus, J.-D. Gascuel, and M. Weinfeld, "Specification and implementation of a digital Hopfield-type associative memory with on-chip training," *IEEE Transactions on Neural Networks* 3 (1992), 529–539.

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

[72] P. Kanerva, *Sparse Distributed Memory.* MIT Press, Cambridge, Massachusetts, 1988.

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

[74] J. D. Keeler, "Basins of attraction of neural network models," in *Proc. Neural Networks for Computing,* ed. J. Denker. AIP Conference Proceedings Vol. 151, American Institute of Physics, New York, 1986, 259–264.

[75] Y. Kobuchi, "State evaluation functions and Lyapunov functions for neural networks," *Neural Networks* 4 (1991), 505–510.

[76] T. Kohonen, "Correlation Matrix Memories," *IEEE Transactions on Computers* 21 (1972), 353–359.

[77] T. Kohonen, *Self-Organization and Associative Memory.* Springer-Verlag, Berlin, 1984; second edition, 1988; third edition, 1989.

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

[79] B. Kosko, "Bidirectional associative memories," *IEEE Transactions on Systems, Man, and Cybernetics* 18 (1988), 49–60.

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

[81] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Magazine,* April 1987, 4–22.

[82] R. P. Lippmann, B. Gold, and M. L. Malpass, *A comparison of Hamming and Hopfield neural nets for pattern classification.* Technical Report 769, MIT Lincoln Laboratories, Lexington, Massachusetts, 1987.

[83] W. A. Little, "The existence of persistent states in the brain," *Mathematical Bioscience* 19 (1974), 101–120.

[84] W. A. Little and G. L. Shaw, "Analytic study of the memory storage capacity of a neural network," *Mathematical Biosciences* 39 (1978), 281–290.

[85] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics* 5 (1943), 115–133.

[86] 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 (1987), 461–482.

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

[88] M. L. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry.* MIT Press, Cambridge, Massachusetts, 1969; expanded edition, 1988.

[89] 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 (1986), 3759–3766.

[90] R. J. T. Morris and W. S. Wong, "A short-term neural network memory," *SIAM Journal on Computing* 17 (1988), 1103–1118.

[91] K. Nakano, "Associatron — A model of associative memory," *IEEE Transactions on Systems, Man, and Cybernetics* 2 (1972), 380–388.

[92] K. Nakano and J.-I. Nagumo, "Information processing using a model of associative memory," in *Proc. Second International Joint Conference on Artificial Intelligence.* University Microfilms, High Wycomb, England, 1971, 101–106.

[93] R. C. Nelson, "Visual homing using an associative memory," *Biological Cybernetics* 65 (1991), 281–291.

[94] J. von Neumann, "First draft of a report on the EDVAC," 1945, reprinted partially in *The Origins of Digital Computers: Selected Papers,* ed. B. Randell, Springer-Verlag, Berlin, Germany, 1973; second edition, 1975; third edition, 1982, 383–392.

[95] J. von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," in *Automata Studies,* eds. C. E. Shannon and J. McCarthy. Princeton University Press, Princeton, New Jersey, 1956, 43–98.

[96] 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.

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

[98] 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, The Netherlands, 1990, 217–268.

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

[100] S. Porat, "Stability and looping in connectionist models with asymmetric weights," *Biological Cybernetics* 60 (1989), 335–344.

[101] P. W. Purdom Jr. and C. A. Brown, *The Analysis of Algorithms.* Holt, Rinehart and Winston, New York, 1985.

[102] M. E. Robinson, H. Yoneda, and E. Sánchez-Sinencio, "A modular CMOS design of a Hamming network," *IEEE Transactions on Neural Networks* 3 (1992), 444–456.

[103] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain," *Psychological Review* 65 (1958), 386–408.

[104] F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* Spartan Books, Washington D.C., 1961.

[105] M. H. Ross, E. J. Reith, and L. J. Romrell, *Histology: A Text and Atlas.* Williams & Wilkins, Baltimore, Maryland, 1985; second edition, 1989.

[106] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1,* eds. D. E. Rumelhart and J. L. McClelland, MIT Press, Cambridge, Massachusetts, 1986.

[107] J. P. Sage, K. Thompson, and R. S. Withers, "An artificial neural network integrated circuit based on MNOS/CCD principles," in *Proc. Neural Networks for Computing,* ed. J. Denker. AIP Conference Proceedings Vol. 151, American Institute of Physics, New York, 1986, 381–385.

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

[109] J. Simon, "On the difference between one and many (preliminary version)," in *Proc. 4th International Conference on Automata, Languages and Programming.* Lecture Notes in Computer Science 52, Springer-Verlag, Berlin, 1977, 480–491.

[110] D. L. Slotnik, E. M. Butterfield, E. S. Colantonio, D. J. Kopetzky, and J. K. Slotnik, *Computers & Applications.* D. C. Heath and Company, Lexington, Massachusetts, first edition, 1986; second edition, 1989.

[111] K. Steinbuch, "Die Lernmatrix," *Kybernetik* 1 (1961), 36–45.

[112] K. Steinbuch and U. A. W. Piske, "Learning matrices and their applications," *IEEE Transactions on Electronic Computers* 1963, 846–862.

[113] H. J. Sussmann, "On the number of memories that can be perfectly stored in a neural net with Hebb weights," *IEEE Transactions on Information Theory* 35 (1989), 174–178.

[114] 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* 10 (1980), 2769–2778.

[115] W. K. Taylor, "Electrical simulation of some nervous system function activities," in *Information Theory*, ed. E. C. Cherry. Butterworths, London, England, 1956, 314–328.

[116] W. K. Taylor, "Cortico-thalamic organization and memory," *Proc. Royal Society of London B* 159 (1964), 466-478.

[117] M. Tchuente, "Sequential simulation of parallel iterations and applications," *Theoretical Computer Science* 48 (1986), 135–144.

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

[119] S. S. Venkatesh, "Robustness in neural computation: random graphs and sparsity," *IEEE Transactions on Information Theory* 38 (1992), 1114–1119.

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

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

[122] J.-H. Wang, T. F. Krile, and J. F. Walkup, "Determination of Hopfield associative memory characteristics using a single parameter," *Neural Networks* 3 (1990), 319–331.

[123] G. Weisbuch and F. Fogelman-Soulié, "Scaling laws for the attractors of Hopfield networks," *Le Journal de Physique — Lettres* 46 (1985), 623–630.

[124] P. J. Werbos, *Beyond regression: New tools for prediction and analysis in the behavioral sciences.* Doctoral dissertation. Applied Mathematics, Harvard University, November 1974.

[125] 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.

[126] J. Wiedermann, "On the computational efficiency of symmetric neural networks," *Theoretical Computer Science* 80 (1991), 337–345.

[127] D. Willshaw, "Holography, associative memory, and inductive generalization," in *Parallel Models of Associative Memory,* eds. G. E. Hinton and S. A. Anderson. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1981; updated edition, 1989, 103–124.

[128] D. J. Willshaw, O. P. Buneman, and H. C. Longuet-Higgins, "Non-holographic associative memory," *Nature (London)* 222 (1969). 960–962.

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

[130] G. J. Woeginger and Z. Yu, "On the equal-subset-sum problem," *Information Processing Letters* 42 (1992), 299–302.

[131] X. Yao, "Finding approximate solutions to NP-hard problems by neural networks is hard," *Information Processing Letters* 41 (1992), 93–98.

I Mathematical Notation

Vectors are denoted by small letters $x, y, z, u, v, \alpha, \beta, \gamma, \xi$:

x_i	i^{th} element of vector x.
x^i	i^{th} x-vector in a collection of x-vectors.
$x(t)$	Vector x at time t when x is changing with time. Initially the vector is $x(0)$.
(x, y)	When $x = (x_1, x_2, \ldots x_m)$ and $y = (y_1, y_2, \ldots y_n)$, then $(x, y) = (x_1, x_2, \ldots x_m, y_1, y_2, \ldots y_n)$.
$\overline{0}$	$\overline{0} = (0, 0, \ldots, 0)$; in the same way $\overline{1} = (1, 1, \ldots, 1)$ and $-\overline{1} = (-1, -1, \ldots, -1)$.
$f(x_1, x_2, \ldots, x_n)$	When f is defined for one argument, we interpret notation $f(x_1, x_2, \ldots, x_n)$ as $(f(x_1), f(x_2), \ldots, f(x_n))$.

Matrices are denoted by capital letters A, B, C, W:

$(A)_{ij}$	The element at row i and column j of matrix A.
a_{ij}	$(A)_{ij}$
(a_{ij})	The matrix consisting of the elements a_{ij}.
A^{T}	Transpose of A: $(A^{\mathrm{T}})_{ij} = a_{ji}$.
I	Identity matrix: $I_{ii} = 1$ for all i, and $I_{ij} = 0$ for all pairs i, j, where $i \neq j$.

Products of vectors:

$x^{\mathrm{T}}y$	Vector product, or inner product, of two vectors of equal length: $x^{\mathrm{T}}y = \sum_j x_j y_j$. This quantity is a scalar.
xy^{T}	Tensor product, or outer product, of two vectors not necessarily of the same length: $xy^{\mathrm{T}} = (x_i y_j)$. This quantity is a matrix.

Metrics:

$d_H(x, y)$	Hamming distance between two binary vectors x and y of equal length, i.e., the number of bits in which the vectors differ from one another. Thus if $x, y \in \{-1, 1\}^n$, then the Hamming distance $d_H(x, y) = (n - x^{\mathrm{T}} y)/2$.
$d_p(x, y)$	Other metrics: $d_p(x, y) = (\sum_j \lvert x_j - y_j \rvert^p)^{1/p}$. The Euclidean metric is thus denoted $d_2(x, y)$. Note especially the 1-metric (sometimes called Manhattan metric) $d_1(x, y) = \sum_j \lvert x_j - y_j \rvert$. If $x, y \in \{-1, 1\}^n$, then $d_H(x, y) = (d_p(x, y)/2)^p$.

O-notation:

$f(n) = O(g(n))$	if there exist constants N and C, such that $f(n) < Cg(n)$ for all $n > N$.
$f(n) = \Omega(g(n))$	if there exists a constant C, such that $f(n) > Cg(n)$ for infinitely many n.
$f(n) = \Theta(g(n))$	if $f(n) = O(g(n))$ and $g(n) = O(f(n))$.
$f(n) = o(g(n))$	if $\lim_{n \to \infty} f(n)/g(n) = 0$.

For other standard mathematical concepts and notation, see any textbooks on calculus and linear algebra.

II Limit of the Probability of Uniqueness

The probability that there is a unique nearest stored vector to the input vector is

$$U(m,n) = \frac{m}{2^{mn}} \sum_{s=1}^{n} \left[\binom{n}{s} \left[\sum_{t=0}^{s-1} \binom{n}{t} \right]^{m-1} \right],$$

as shown in Section 5.2. We study the limit of $U(m(n),n)$ when $m(n)$ is a function of n. The results in this appendix are joint work with Taneli Huuskonen [36].

Denote $x_j = \sum_{i=0}^{j} \binom{n}{i}/2^n$. Now $x_0 = 1/2^n$, $x_n = 1$, and

$$U(m,n) = m \sum_{j=1}^{n} (x_j - x_{j-1}) x_{j-1}^{m-1} \leq m \int_{x_0}^{x_n} x^{m-1}\, dx = 1 - 2^{-mn},$$

because the sum is a lower sum for the Riemann integral.

With higher m, the maximum difference of $x_k^{m-1} - x_{k-1}^{m-1}$ grows bigger, in general, so the sum $m \sum_{k=1}^{n} (x_k - x_{k-1}) x_{k-1}^{m-1}$ differs more and more from the integral. However, with higher n, the sum comes closer to the integral. How fast can m grow as a function of n, so that the growth of m is compensated by the growth of n and the limit probability is 1? We show in the following that $\lim_{n\to\infty} U(m(n),n) = 1$ when $m(n)$ is a subexponential function of n, i.e., $\log m(n) = o(n)$.

We begin with some lemmas.

Lemma A – 1 *If $j,k \geq 0$ and $j+k \leq n$, then*

$$\left(\frac{n-j-k+1}{j+k}\right)^k \leq \binom{n}{j+k} \Big/ \binom{n}{j} \leq \left(\frac{n-j}{j+1}\right)^k.$$

Proof Clearly, $\binom{n}{i+1}/\binom{n}{i} = (n-i)/(i+1)$, and this decreases with i. Now

$$\binom{n}{j+k} \Big/ \binom{n}{j} = \prod_{i=j}^{j+k-1} \binom{n}{i+1} \Big/ \binom{n}{i} = \prod_{i=j}^{j+k-1} \frac{n-i}{i+1} \leq \prod_{i=j}^{j+k-1} \frac{n-j}{j+1} = \left(\frac{n-j}{j+1}\right)^k.$$

Analogously,

$$\binom{n}{j+k} \Big/ \binom{n}{j} = \prod_{i=j}^{j+k-1} \binom{n}{i+1} \Big/ \binom{n}{i} \geq \left(\frac{n-j-k+1}{j+k}\right)^k.$$

□

Lemma A – 2 *Choose* $\delta \in]0, \frac{1}{30}[$ *and* $n \geq 5$. *Then*

$$\sum_{j=\lceil n/2+\delta n\rceil+1}^{n} (x_j - x_{j-1})^2 x_j^{m-2} < e^{-4\delta^2 n}.$$

Proof Denote

$$S = \sum_{j=\lceil n/2+\delta n\rceil+1}^{n} (x_j - x_{j-1})^2 x_j^{m-2}.$$

Note that $\lceil n/2 + \delta n\rceil + 1 < n$, when $\delta < 1/30$ and $n \geq 5$. Clearly,

$$\begin{aligned} S &< \sum_{j=\lceil n/2+\delta n\rceil+1}^{n} (x_j - x_{j-1})^2 \\ &< \left(\sum_{j=\lceil n/2+\delta n\rceil+1}^{n} (x_j - x_{j-1})\right)^2 = \left(2^{-n} \sum_{j=\lceil n/2+\delta n\rceil+1}^{n} \binom{n}{j}\right)^2. \end{aligned}$$

But this is nothing else than the square of a tail of a binomial distribution, for which we can use a Hoeffding bound (see Appendix C in [63]), and we get

$$S < e^{-4\delta^2 n}.$$

□

Lemma A – 3 *Let* $\delta \in \left]0, \frac{1}{30}\right[$. *If* $j \leq \lceil n/2 + \delta n\rceil$, $n \geq 12/\delta$, *and* $\left(\frac{1-4\delta}{1+4\delta}\right)^{\lceil \delta n\rceil} < 1/100$, *then* $x_j - x_{j-1} < 10\delta(1 - x_j)$.

Proof Note that $n \geq 12/\delta > 360$, and hence $j+\delta n \leq \lceil n/2+\delta n\rceil + \lceil \delta n\rceil < 17n/30+2 < n$. Thus

$$\frac{x_j - x_{j-1}}{1 - x_j} = \frac{\binom{n}{j}}{\sum_{k=j+1}^{n} \binom{n}{k}} < \frac{\binom{n}{j}}{\sum_{k=j+1}^{j+\lceil \delta n\rceil} \binom{n}{k}} = \frac{\binom{n}{j}}{\sum_{k=1}^{\lceil \delta n\rceil} \binom{n}{k+j}}.$$

According to Lemma A – 1,

$$\binom{n}{j+k} \geq \binom{n}{j} \left(\frac{n-j-k+1}{j+k}\right)^k \geq \binom{n}{j} \left(\frac{n - \lceil n/2 + \delta n\rceil - \lceil \delta n\rceil + 1}{\lceil n/2 + \delta n\rceil + \lceil \delta n\rceil}\right)^k,$$

because $j + k \leq \lceil n/2 + \delta n\rceil + \lceil \delta n\rceil$. But

$$\frac{n - \lceil n/2 + \delta n\rceil - \lceil \delta n\rceil + 1}{\lceil n/2 + \delta n\rceil + \lceil \delta n\rceil} \geq \frac{n - n/2 - 2\delta n - 1}{n/2 + 2\delta n + 2} = \frac{1 - 4\delta - 2/n}{1 + 4\delta + 4/n}.$$

From the assumptions it follows that

$$\frac{8\delta + 6/n}{1 - 4\delta - 2/n} < \frac{306}{31}\delta,$$

so the ratio $(x_j - x_{j-1})/(1 - x_j)$ is bounded from above by

$$\begin{aligned}
\frac{1}{\sum_{k=1}^{\lceil \delta n \rceil} \left(\frac{1-4\delta-2/n}{1+4\delta+4/n}\right)^k} &= \frac{1}{1 - \left(\frac{1-4\delta-2/n}{1+4\delta+4/n}\right)^{\lceil \delta n \rceil}} \left(\frac{1+4\delta+4/n}{1-4\delta-2/n} - 1\right) \\
&< \frac{1}{1 - 1/100}\left(\frac{8\delta + 6/n}{1 - 4\delta - 2/n}\right) \\
&< \frac{100}{99}\frac{306}{31}\delta < 10\delta.
\end{aligned}$$

□

Lemma A − 4 *Assume that* $x_{\lceil n/2 \rceil - 1} \geq 1/3$. *Choose* $\delta \in]0, \frac{1}{2}]$. *If there is a constant* $c < 1/3$ *such that* $x_j - x_{j-1} \leq c(1 - x_j)$ *for all* $\lceil n/2 \rceil \leq j \leq \lceil n/2 + \delta n \rceil$, *then*

$$\binom{m}{2} \sum_{j=\lceil n/2 \rceil}^{\lceil n/2 + \delta n \rceil} (x_j - x_{j-1})^2 x_j^{m-2} < \frac{c}{2(1 - 3c)^2}.$$

Proof Denote

$$S = \binom{m}{2} \sum_{j=\lceil n/2 \rceil}^{\lceil n/2 + \delta n \rceil} (x_j - x_{j-1})^2 x_j^{m-2} \leq \binom{m}{2} c \sum_{j=\lceil n/2 \rceil}^{\lceil n/2 + \delta n \rceil} x_j^{m-2}(1 - x_j) \int_{x_{j-1}}^{x_j} dx.$$

But $1 - x_j \leq 1 - x$ when $x \in [x_{j-1}, x_j]$, so

$$S \leq \binom{m}{2} c \sum_{j=\lceil n/2 \rceil}^{\lceil n/2 + \delta n \rceil} x_j^{m-2} \int_{x_{j-1}}^{x_j} (1 - x)\, dx.$$

Now $x_j^{m-2} \leq x^{m-2}(x_j/x_{j-1})^{m-2}$ when $x \in [x_{j-1}, x_j]$, and

$$x_j/x_{j-1} = 1 + (x_j - x_{j-1})/x_{j-1} \leq 1 + c(1 - x_j)/x_{j-1} \leq 1 + 3c(1 - x_j),$$

as $x_{j-1} \geq 1/3$. Consequently, we get the bound $x_j^{m-2} \leq x^{m-2}(1 + 3c(1 - x))^{m-2}$ when $x \in [x_{j-1}, x_j]$. Thus,

$$S \leq \binom{m}{2} c \int_{x_{\lceil n/2 \rceil - 1}}^{x_{\lceil n/2 + \delta n \rceil}} x^{m-2}(1 + 3c(1 - x))^{m-2}(1 - x)\, dx$$

$$
\begin{aligned}
&< \binom{m}{2} c \int_0^1 x^{m-2}(1+3c(1-x))^{m-2}(1-x)\,dx \\
&= \binom{m}{2} c \int_0^1 (1+(3c-1)(1-x)-3c(1-x)^2)^{m-2}(1-x)\,dx \\
&< \binom{m}{2} c \int_0^1 (1+(3c-1)(1-x))^{m-2}(1-x)\,dx.
\end{aligned}
$$

Substituting y for $1+(3c-1)(1-x)$ we get

$$
\begin{aligned}
S &< \binom{m}{2} \frac{1}{(1-3c)^2} \int_{3c}^1 y^{m-2}(1-y)\,dy \\
&= \binom{m}{2} \frac{c}{(1-3c)^2} \left(\frac{1}{m(m-1)} + \frac{(3c)^m}{m} - \frac{(3c)^{m-1}}{m-1} \right) \\
&< \frac{c}{(1-3c)^2} \binom{m}{2} \frac{1}{m(m-1)} = \frac{c}{2(1-3c)^2}.
\end{aligned}
$$

□

Theorem A – 1 *Assume that $m(n)$ is subexponential. Then* $\lim_{n\to\infty} U(m(n),n) = 1$.

Proof In the following, we denote $m = m(n)$. The probability that two of the random values are both equal to j and all the rest are at most j is $(x_j - x_{j-1})^2 x_j^{m-2}$. There are $\binom{m}{2}$ such pairs; thus the probability of the maximum being equal to j and not unique is at most $\binom{m}{2} \sum_{j=0}^n (x_j - x_{j-1})^2 x_j^{m-2}$. Hence,

$$
0 \le 1 - U(m,n) \le \binom{m}{2} \sum_{j=0}^{n} (x_j - x_{j-1})^2 x_j^{m-2} \le 2\binom{m}{2} \sum_{j=\lceil n/2 \rceil}^{n} (x_j - x_{j-1})^2 x_j^{m-2}.
$$

Let us denote the last expression by $U'(m,n)$. We now proceed to prove that the limit of U' is zero, thus forcing the limit of U to be 1.

Let us divide the sum defining $U'(m,n)$ into two parts: $j = \lceil n/2 \rceil, \ldots, \lceil n/2 + \delta n \rceil$, and $j = \lceil n/2 + \delta n \rceil + 1, \ldots, n$, where $\delta = \delta(n) = \sqrt{\ln(mn)/(2n)}$. Now, since m is subexponential as a function of n, there is an n_0 such that $\delta(n) < 1/60$ for all $n \ge n_0$. Obviously, $n_0 \ge 1/2\delta(n_0)^2 > 1800$. Assume from now on that $n \ge n_0$.

We study now the assumptions for Lemma A – 3. From the series expansion of the expression $\ln(1-x)/\ln(1+x)$ and from the equality $\delta^2 n = \ln(mn)/2$, we get the bound

$$\left(\frac{1-4\delta}{1+4\delta}\right)^{\lceil \delta n \rceil} < e^{-8\delta^2 n} = (mn)^{-4} < \frac{1}{100}.$$

It is easy to see that also the other assumptions are satisfied, so Lemma A – 3 implies that $x_j - x_{j-1} < 10\delta(1-x_j)$ for $j \leq \lceil n/2 + \delta n \rceil$. In other words, we choose $c = 10\delta$ in Lemma A – 4.

As $x_{\lceil n/2 \rceil} \geq 1/2$ and $x_{\lceil n/2 \rceil} - x_{\lceil n/2 \rceil - 1} < 10\delta < 1/6$, we have $x_{\lceil n/2 \rceil - 1} \geq 1/3$. Thus the premises for Lemma A – 4 hold and

$$\binom{m}{2} \sum_{j=\lceil n/2 \rceil}^{\lceil n/2+\delta n \rceil} (x_j - x_{j-1})^2 x_j^{m-2} < \frac{10\delta}{2(1-30\delta)^2} < 20\delta.$$

Also, the premises for Lemma A – 2 hold and using $\delta^2 n = \ln(mn)/2$ we get

$$\binom{m}{2} \sum_{j=\lceil n/2+\delta n \rceil + 1}^{n} (x_j - x_{j-1})^2 x_j^{m-2} < \binom{m}{2} e^{-4\delta^2 n} < \frac{1}{2n^2} \leq 2\delta^4.$$

Consequently,

$$U'(m,n) < 40\delta + 4\delta^4 < 41\delta = 41\sqrt{\ln(mn)/(2n)} \to 0,$$

as $n \to \infty$ and $\log m = o(n)$. $\square$

III The Calculator Test Set

The 40 stored vectors in the test described in Section 6.3 are shown below. Each vector has 35 elements, and in the figure the elements are grouped in 7 rows of 5 elements each. The vectors were taken from the display of a calculator and consist of the ten digits, all the 26 alphabetic characters, and the symbols "+", "−", "*", and "/". In the figure below, ■ means value +1, and - means value −1.

```
-■■■-  --■--  -■■■-  ■■■■■  ---■-  ■■■■■  --■■-  ■■■■■  -■■■-  -■■■-
■---■  -■■--  ■---■  ---■-  --■■-  ■----  -■---  ----■  ■---■  ■---■
■--■■  --■--  ----■  --■--  -■-■-  ■■■■-  ■----  ---■-  ■---■  ■---■
■-■-■  --■--  ---■-  ---■-  ■--■-  ----■  ■■■■-  --■--  -■■■-  -■■■■
■■--■  --■--  --■--  ----■  ■■■■■  ----■  ■---■  --■--  ■---■  ----■
■---■  --■--  -■---  ■---■  ---■-  ■---■  ■---■  --■--  ■---■  ---■-
-■■■-  -■■■-  ■■■■■  -■■■-  ---■-  -■■■-  -■■■-  --■--  -■■■-  -■■--

-■■■-  ■■■■-  -■■■-  ■■■--  ■■■■■  ■■■■■  -■■■-  ■---■  -■■■-  --■■■
■---■  ■---■  ■---■  ■--■-  ■----  ■----  ■---■  ■---■  --■--  ---■-
■---■  ■---■  ■----  ■---■  ■----  ■----  ■----  ■---■  --■--  ---■-
■■■■■  ■■■■-  ■----  ■---■  ■■■■-  ■■■■-  ■-■■■  ■■■■■  --■--  ---■-
■---■  ■---■  ■----  ■---■  ■----  ■----  ■---■  ■---■  --■--  ---■-
■---■  ■---■  ■---■  ■--■-  ■----  ■----  ■---■  ■---■  --■--  ■--■-
■---■  ■■■■-  -■■■-  ■■■--  ■■■■■  ■----  -■■■■  ■---■  -■■■-  -■■--

■---■  ■----  ■---■  ■---■  -■■■-  ■■■■-  -■■■-  ■■■■-  -■■■-  ■■■■■
■--■-  ■----  ■■-■■  ■---■  ■---■  ■---■  ■---■  ■---■  ■---■  --■--
■-■--  ■----  ■-■-■  ■■--■  ■---■  ■---■  ■---■  ■---■  ■----  --■--
■■---  ■----  ■-■-■  ■-■-■  ■---■  ■■■■-  ■---■  ■■■■-  -■■■-  --■--
■-■--  ■----  ■---■  ■--■■  ■---■  ■----  ■-■-■  ■-■--  ----■  --■--
■--■-  ■----  ■---■  ■---■  ■---■  ■----  ■--■-  ■--■-  ■---■  --■--
■---■  ■■■■■  ■---■  ■---■  -■■■-  ■----  -■■-■  ■---■  -■■■-  --■--

■---■  ■---■  ■---■  ■---■  ■---■  ■■■■■  -----  -----  -----  -----
■---■  ■---■  ■---■  ■---■  ■---■  ----■  --■--  -----  --■--  ----■
■---■  ■---■  ■---■  -■-■-  -■-■-  ---■-  --■--  -----  ■-■-■  ---■-
■---■  ■---■  ■-■-■  --■--  --■--  --■--  ■■■■■  ■■■■■  -■■■-  --■--
■---■  ■---■  ■-■-■  -■-■-  --■--  -■---  --■--  -----  ■-■-■  -■---
■---■  -■-■-  ■■-■■  ■---■  --■--  ■----  --■--  -----  --■--  ■----
-■■■-  --■--  ■---■  ■---■  --■--  ■■■■■  -----  -----  -----  -----
```