Complexity Issues in Discrete Neurocomputing

by

Juraj Wiedermannn *

A 10/90

May 1990

FB 10, Informatik
Universität des Saarlandes
D–6600 Saarbrücken
West Germany

Abstract: An overview of the basic results in complexity theory of discrete neural computations is presented. Especially, the computational power and efficiency of single neurons, neural circuits, symmetric neural networks (Hopfield model), and of Boltzmann machines is investigated and characterized. Corresponding intractability results are mentioned as well. The evidence is presented why discrete neural networks (inclusively Boltzmann machines) are not to be expected to solve intractable problems more efficiently than other conventional models of computing.

This work was finished while the author was on leave from VUSEI-AR, Bratislava, ČSFR. The research was partially supported by the ESPRIT II Basic Research Actions Program of the EC under contract No. 3075 (Project ALCOM).

Complexity Issues in Discrete Neurocomputing

Juraj Wiedermann
VUSEI-AR, Dúbravská 3, 842 21 Bratislava
Czechoslovakia

Abstract: An overview of the basic results in complexity theory of discrete neural computations is presented. Especially, the computational power and efficiency of single neurons, neural circuits, symmetric neural networks (Hopfield model), and of Boltzmann machines is investigated and characterized. Corresponding intractability results are mentioned as well. The evidence is presented why discrete neural networks (inclusively Boltzmann machines) are not to be expected to solve intractable problems more efficiently than other conventional models of computing.

1. Introduction

1.1. *Motivation.* The recent renewed interest in neurocomputing is undoubtedly motivated by our ever increasing quest for exploiting new, non-traditional ways of computing. Along these lines at the border between computational physics and neurobiology a new computational paradigm is emerging saying that certain collective spontaneous properties of a mass of some simple computational devices can be used to immediately realize the computations. This gives rise to a brand-new class of computational machines in which the physics of the machine is intimately related to the algorithm of computations.

The prominent representatives of simple computational devices from which the resulting machines are assembled, are (artificial) neurons. These neurons are connected into a neural network and depending on the way in which the neurons work and cooperate, and on the topology of the resulting network, various types of neural nets can be distinguished: neural circuits, symmetric neural networks (so-called Hopfield neural networks), Boltzmann machines, etc.

So far these machines have been experimentally used for solving various isolated problems, like associative memory realizations [H1, H2], solving some combinatorial problems [AH,HT1,KGV] , simple models of learning [AH], or speech recognition [H3].

Despite some promising experimental evidence of these machines no sufficient attention from the side of computer science has been paid to these machines and therefore

This work was finished while the author was visiting the Department of Computer Science, University of Saarland, West Germany (Spring 1990)

the corresponding complexity theory that would answer the general questions concerning their computational power and efficiency has emerged only slowly and in fact only recently [P, W1].

One reason for this unfortunate state of the matters could be that until recently the corresponding devices have developed themselves mostly outside the framework of complexity theory — viz. within the framework of artificial intelligence, computational physics, or neurobiology. As a result, the main emphasis was put mostly on the learning abilities of these devices, too often without bothering explicitly about their computational power and efficiency. The second reason could have been that the results were scattered throughout non-computer science journals, the respective models varied from author to author, the used terminology ranged from that of mechanics through electricity up to biology, both discrete as well as analog devices were used, and therefore it was difficult for an 'outsider' to arrive at a reasonable computational model, from the viewpoint of complexity theory.

But, slowly, as the field has matured and certain devices have established themselves as more or less fundamental ones, the attention of computer science has been caught, especially when the neurocomputing researchers reported unusual efficiency of their devices in solving some intractable problems [HT1, J, KA].

Nowadays, at least as far as discrete neurocomputing is concerned, the complexity theory disposes of quite a solid body of knowledge about these devices that places them into the proper perspective among the other known models of computations and thus explains some of experimentally observed phenomena [W2].

Nevertheless, the analog neurocomputing remains still outside the reach of today's complexity theory, as it is regrettably the case with analog computations per se.

1.2. *The aims.* The goal of the paper presented is to give a contemporary state-of-the art survey on the basic complexity results concerning the fundamental classes of discrete neural machine models. We shall not be concerned in neural learning although this topic traditionally ranks among prime 'practical' motivations in studying neurocomputing. Notwithstanding, our results will shed some light on a problem what, at least in principle, can be learned by neural nets, and therefore they can be seen as a prelude to more advanced studies in neural learning. As a preliminary or companion reading an excellent introduction by Parberry [P] is recommended. The present paper tries to complement the Parberry's paper by new results or by results not covered in details in his paper. However, in striving for selfconsistency of our paper the overlapping of some topics could not have been avoided.

1.3. *Contents.* First, in Section 2, we shall introduce the notion of an abstract neuron and we shall characterize its computational power. We shall also present a new result that such an apparently simple problem as deciding whether a given boolean function can be realized by a single neuron, presents a Σ_2–complete problem (where Σ_2 denotes the complexity class in Stockmeyer's polynomial time hierarchy — see e.g. [CKS] or [GJ]). This result explains the exponential complexity of all known neural learning algorithms, most notably of that by Rosenblatt (which is known as perceptron learning procedure [MP]).

In Section 3 we shall introduce the notion of neural networks and describe the way they compute, and the corresponding complexity measures.

In Section 4 the notion of neural circuits will be introduced and we shall briefly investigate their computational power and efficiency. We shall show that neural circuits can simulate efficiently any neural network. Moreover we shall also show that in the case of bounded fan-in these circuits are equivalent to standard combinatorial circuits, while in the case of unbounded fan-in they can compute any boolean function in constant parallel time, however, using an exponential number of neurons. Therefore in the rest of this Section we shall focus our attention on an interesting intermediate case of neural circuits of polynomial size and constant depth.

In Section 5 we shall continue our excursion with investigation of symmetric neural nets (so-called Hopfield model [H2]). We shall show that the computational power and efficiency of these networks is equivalent to that of neural circuits [W1, W2]. Further we shall introduce the notion of energy function for symmetric neural networks which exemplifies the close connection between computations of these networks and energy states of certain physical systems. We shall state here the fundamental Hopfield's result that any computation of symmetric neural networks can be seen as the minimization process of the corresponding energy function [H2]. We will also study the relation between nondeterministic computations and energy function minimization problem and we show that the process of minimizing the energy function of a symmetric neural network presents an NP–complete problem. As a consequence we obtain the main result of this Section stating that any nondeterministic Turing machine computation can be realized by a symmetric neural network that ends its computation in a state that presents the global minimum of the corresponding energy function [W2]. We shall close this section by investigating briefly the terminating problem for parallel symmetric neural networks.

Finally, in Section 6 we shall further enrich the computational capabilities of neural networks by introducing the probability into the computations of the respective neurons. As a result we obtain a discrete Boltzmann machine, studied intensively in the connection with so-called simulated annealing [KGV]. We will sketch here the recent surprising result by Parberry and Schnitger [PS] that from computational complexity point of view these machines are equivalent to neural circuits introduced in Section 4.

In Conclusions we shall briefly summarize the importance and the contribution of neurocomputing to complexity theory.

2. Neurons

2.1. *Basic definitions.* A 'neuron' is a catchy name, inspired by the analogy with real neurons (in biology), of an abstract device that is capable to compute the values of so-called linearly separable, or weighted threshold functions in one computational step.

Definition 2.1. *A boolean function f of n variables $\vec{x} = (x_1, x_2, \ldots, x_n)$ is called a linearly separable function if and only if there is an integer vector $\vec{w} = (w_1, w_2, \ldots, w_n)$ and an integer constant t such that the set $f^{-1}(1) \overset{def}{=} \{\vec{x} \mid f(\vec{x}) = 1\}$ equals the set $\{\vec{x} \mid (\vec{w}, \vec{x}) \geq t\}$ and $f^{-1}(0) \overset{def}{=} \{\vec{y} \mid f(\vec{y}) = 0\}$ equals the set $\{\vec{y} \mid (\vec{w}, \vec{y}) < t\}$, where $(\vec{w}, \vec{x}) = \sum_{i=1}^{n} w_i x_i$ denotes the usual operation of scalar product of vectors $\vec{w}$ and $\vec{x}$.* ∎

Following the qeometrical interpretation we say that the sets $f^{-1}(1)$ and $f^{-1}(0)$ are *linearly separable* by a *separating hyperplane* whose equation in an n-dimensional Euclidean space is given by $(\vec{w}, \vec{x}) = t$.

Such a separable function can be formally represented by a so-called *weighted boolean threshold function* $f[\vec{w}, t]$ defined as follows: $f[\vec{w}, t](\vec{x}) = 1$ iff $(\vec{w}, \vec{x}) \geq t$.

Definition 2.2. *An n-input neuron with weights $(w_1, \ldots, w_n)$ and the threshold t is an abstract device capable to compute the values of a weighted threshold function $f[\vec{w}, t](\vec{x})$ in one computational step, for any $\vec{x}$.* ∎

Schematically, a neuron computing the function $f[\vec{w}, t]$ is depicted in Fig. 1.

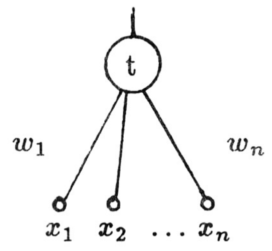

Fig. 1 A neuron

The values of $\vec{x}$ are called *the inputs* of a neuron. When for a given input $\vec{x}$ $f[\vec{w}, t](\vec{x}) = 1$ we say that the corresponding neuron is *active*, or is in an *active state*; its *output* is then defined to be 1; otherwise it is in a *passive state* and its output is then 0.

2.2. *Size of neural representation.* Clearly, by changing the weights or the threshold of a neuron we obtain another neuron that may compute another weighted boolean threshold function. We may now ask what is the number $N(n)$ of different n-input neurons (i.e., those computing different functions)?

Although $N(n)$ is not known to be determined exactly, except for small values of $n \leq 8$ [MTB], its asymptotic behavior is known: from [Mu] it follows that $N(n) = 2^{\Theta(n^2)}$, i.e., there exist approximatively 2^{n^2} different n-input neurons.

This means that to be able to describe an arbitrary n-input neuron in general $\Theta(n^2)$ bits are necessary and enough. It follows that in the worst case there must exist weights of size at least $\Omega(n)$ bits. Futher, it is known [Mu] that it is enough for weights to be of size $O(n \log n)$, and as Parberry [P] noticed it is an open problem whether $O(n)$ bits would be always sufficient.

2.3. *The complexity of finding neural representation.* Given a boolean function f, e.g. by the help of a formula in a conjunctive normal form, it seems that it is not an easy task to find the corresponding neural representation (i.e., its representation as a weighted boolean threshold function), providing that f is a linearly separable function. In fact, the next theorem shows that the corresponding decision problem of asking whether a given boolean function is linearly separable presents an intractable problem.

Theorem 2.1. *The SEPARABILITY problem of deciding whether a given boolean function f is linearly separable is a Σ_2-complete problem.*

Proof (Sketch): First we shall show that the above problem is in Σ_2, i.e., in the class of polynomially time–bounded alternating Turing machine computations that use at most 2 alternations on each computational path, starting in an existential state (see [CKS] and [GJ] for the definition of the complexity class Σ_2).

Consider therefore an alternating Turing machine M that works in the following manner. The machine M starts in existential mode guessing the neural representation f_N of f — which is of size $O(n^2 \log n)$ bits (see 2.2), and therefore can be guessed in polynomial time. Then, in parallel, M verifies for each input $\vec{x}$ of size n whether $f_N(\vec{x}) = f(\vec{x})$. This takes again a polynomial time.

Secondly, we show that the problem of testing the validity of a given quantified boolean formula F in a conjunctive normal form with m variables, starting with existential quantifiers followed by universal ones, is polynomial-time reducible to SEPARABILITY. Since it is known that the validity problem of such formulae presents a Σ_2-complete problem [GJ], this will be enough to prove the Σ_2-completeness of SEPARABILITY.

Hence our goal will be to construct a boolean function $f \not\equiv 0$ which will be separable iff F is valid.

W.l.o.g. suppose that F is of form $(\exists x_1)\ldots(\exists x_k)(\forall x_{k+1})\ldots(\forall x_m)g(x_1,\ldots,x_m)$ of length n, where g is a boolean formula, and suppose that F is valid. This means that there exist $z_1,\ldots,z_k \in \{0,1\}$ such that $g(z_1,\ldots,z_k,x_{k+1},\ldots,x_m) = 1$ irrespective of the values of x_i's. W.l.o.g. suppose further that $z_1 = \ldots = z_p = 1$ and $z_{p+1} = \ldots = z_k = 0$, for some p, $0 \le p \le k$, and put $w_1 = \ldots = w_p = 1$, $w_{p+1} = \ldots = w_k = -(p+1)$, $w_{k+1} = \ldots = w_m = 0$, and $t = p$, and define f as $f(\vec{x}) :=$ **if** $(\vec{w},\vec{x}) \ge t$ **then** $g(\vec{x})$ **else** 0 **fi**.

We shall show that f is separable.

For that purpose consider the values of $(\vec{w},\vec{x})$ for all $\vec{x}$. If the vector $\vec{x} = (z_1,\ldots,z_k, x_{k+1},\ldots,x_m)$, then $(\vec{w},\vec{x}) \ge t$, $g(\vec{x}) = 1$, and therefore also $f(\vec{x}) = 1$. On the other hand, when the first k components of $\vec{x}$ are different from $z_1,\ldots,z_k$, the scalar product $(\vec{w},\vec{x}) < t$, and due to the definition of f, $f(\vec{x}) = 0$. This means that $h : (\vec{w},\vec{x}) = t$ is a separating hyperplane of f indeed, and therefore f is a separable function.

Let f be a separable function — i.e., there is a hyperplane $h : (\vec{w},\vec{x}) = t$ such that $f(\vec{x}) = 1$ iff $(\vec{w},\vec{x}) \ge t$. W.l.o.g. suppose that $w_i \ne 0$ for $i = 1,\ldots,k$, and $w_j = 0$ for $j = k+1,\ldots,m$, for some k, $1 \le k \le m$. Then, clearly, the formula $F : (\exists x_1)\ldots(\exists x_k)(\forall x_{k+1})\ldots(\forall x_m)f(x_1,\ldots,x_m)$ is valid: namely, to achieve $f(\vec{x}) = 1$ it is enough to choose such an $\vec{x} : (\vec{w},\vec{x}) \ge t$. ∎

The previous Theorem thus explains the exponential time complexity of all known neuron learning algorithms, among them most notably of that by Rosenblatt (see e.g. in [MP], a so-called perceptron learning procedure).

3. Neural networks

3.1. *Basic definition.* Informally, neural networks are obtained from individual neurons by connecting the outputs of some neurons to inputs of some neurons and by declaring certain neurons as input ones, others as output ones.

Definition 3.1. *A neural network is a 7-tuple* $M = (N, C, I, O, A, w, h)$ *, where*

— N *is a finite set of neurons*
— $C \subseteq N \times N$ *is a set of oriented interconnections among neurons*
— $I \subseteq N$ *is a set of input neurons*
— $O \subseteq N$ *is a set of output neurons*
— $A \subseteq N$ *is a set of initially active neurons,* $A \cap T = \emptyset$
— $w : C \to Z$ *is a weight function;* Z *is the set of all integers*
— $h : N \to Z$ *is a threshold function*

The ordered pair (N, C) *forms an oriented graph that is called an interconnection graph of* M. ∎

Each neuron $u_i \in N$ can enter two different *states*: 0 (inactive) or 1 (active) as characterized by its *output* x_i. There is a so-called *threshold value* $t_i \in Z$ assigned by the function h to each neuron u_i. Each neuron has an arbitrary number of input and output *connections* that are labeled by *weights.* The *total input* to each neuron u_i at any moment is given by the sum $h_i = \sum_{j=1}^{n} a_{i,j} x_j$, where $a_{i,j} \in Z$ is the weight (assigned by the function w) of the u_i's input connection leading from u_j to u_i, x_j is the state of u_j at a given moment and n is the total number of neurons in the network.

3.2. *Neural network computation.* The *computation* of such a system on a given input starts by initializing the states of input neurons in the set I to corresponding *input values* (0 or 1), the states of neurons (if any) in the set A of initialized neurons to 1's, and the remaining neurons are left in a passive state.

The description of states of all neurons in the network at any moment is called a *configuration* of that network at that moment. Further the network can compute in two different ways: in a parallel, and in a sequential mode. In a *parallel (sequential)* mode each neuron u_i samples its inputs synchronously in parallel with other neurons (sequentially, in any order in each step) and if $h_i \geq t_i$ the output x_i is set to 1, otherwise to 0.

The time interval within which the actions of all neurons are accomplished is called a *computational cycle.* Note that in a parallel computational mode the number of computational cycles performed during a computation corresponds to parallel computational time.

The network then works as described above and the computation on a given input is *finished* when a *stable state* is achieved which is the situation in which the state of each neuron remains unchanged during one computational cycle. In that case we say that the computation was *convergent.*

The *result* of the convergent computation on a given input is given by the states of some selected *output neurons.* When the stable state is not reached the output of the computation is not defined.

Note that due to the fact that the neurons in a sequential computation mode can sample their inputs in any order even from the same initial configuration each sequential computation can lead to different results or some (in any mode) can lead to no results at all. It will be our concern to design the network in such a way that the computations will be convergent and the results will be unique if necessary. However, the satisfaction of the latter two conditions can be guaranteed only for certain special classes of neural networks that we shall deal with in a sequel.

3.3. *Complexity measures.* The *time complexity* of a convergent computation on an input of length n will be given as the maximum number of computational cycles needed for achieving a stable state taken over all inputs of length n.

The *size* of the network will be given by the number of its neurons.

4. Neural circuits

4.1. *Basic definition.* Neural circuits are special instances of neural networks:

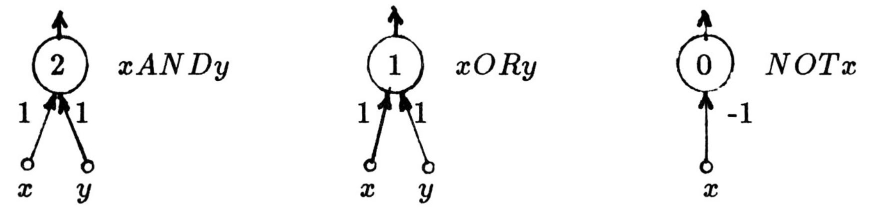

Fig. 2 Neural realization of basic boolean operations

Definition 4.1. *A neural circuit is a neural network with a directed acyclic interconnection pattern; its input neurons are those having no predecessors and its output neurons those having no successors in its interconnection graph.*

The depth of a neural circuit is the length of the longest path between some input and some output neuron. ■

From this definition it follows that neural circuits possess some desirable properties which cannot be guaranteed in the general case of neural networks: it is clear that starting from a given initial configuration any computation of a neural circuit must terminate always in the same stable final configuration, in parallel mode in a time that is proportional to the depth of the circuit, and in sequential mode in a time that is proportional to its size.

4.2. *Neural networks and neural circuits.* Neural circuits, being a special case of neural networks, cannot be computationally more powerfull than neural networks. Nevertheless the following adaptation of a standard simulation technique from complexity theory (see e.g. [PS1, PS2]) shows that everything what can be computed by neural networks can be computed by neural circuits as well, even in the same parallel time.

Theorem 4.1. *Any neural network N of size $S(n)$ and of parallel time complexity $T(n)$ can be simulated by a neural circuit C of size $O(S(n)T(n))$ in parallel time $O(T(n))$.*

Proof (Sketch). We shall construct a circuit C in which the computation proceeds through a sequence of 'layers', $i-$th layer corresponding to i-th configuration c_i of N.

In the first layer of C there is exactly $S(n)$ input neurons. At the beginning of the computation the state of each of them corresponds to that in the initial cinfiguration c_0 of the respective neurons in N. In the next layers inputs to any neuron u in the $(i+1)-$st layer are connected by oriented edges with the same weights as in N to those neurons in the $i-$th layer whose outputs are sampled by u in N, for $i = 0, 1, \ldots, T(n) - 1$. This construction quarantees that there are no cycles in C and when the computation of C terminates the states of neurons in its $T(n)-$th layer correspond to those in the final configuration of N. ■

4.3. *The computational power of neural circuits.* The previous result states that the computational power of neural circuits equals to that of neural networks. But what is the computational power of these devices when compared to more traditional models of computation? It appears that in this respect there is no difference between neural circuits and boolean combinatorial circuits.

It is not difficult to verify that neurons from Fig. 2 realize the basic logical operations AND, OR and NOT; it follows that every combinatorial circuit can be simulated by a neural circuit of the same topology, i.e., of the same size and depth (see also [M] for the proof that any deterministic Turing machine can be build out of neurons!).

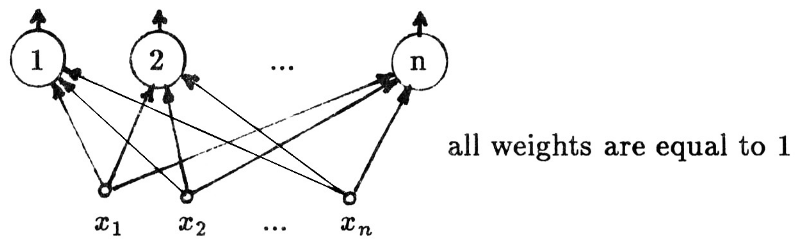

Fig. 4 A neural binary sorter

However, using the 'full capacity' of neural circuits, i.e., the unbounded fan-in, it is possible to compute any boolean function f by a neural circuit of constant depth, simply by constructing a circuit that mimics the representation of f in its disjunctive normal form (Fig. 3).

It follows that in a parallel computation mode the value of any boolean function can be computed by a neural circuit in constant time! It is not known whether in general the size of such circuit can be substantially less than as in the above example while maintaining the constant depth of a circuit.

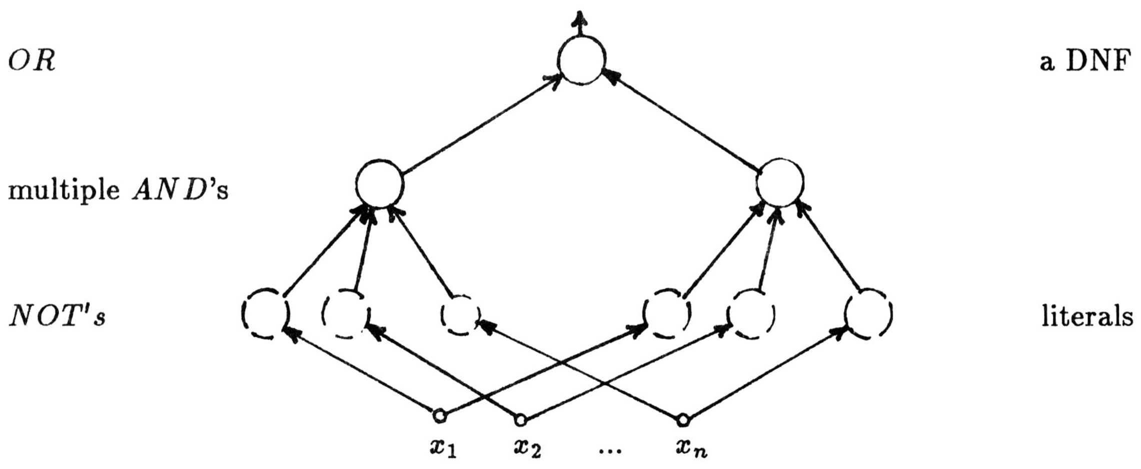

Fig. 3 A circuit computing a boolean function in DNF

4.4. *Neural circuits of constant depth and polynomial size.* The circuit from the last example has the size exponential w.r.t. the number of its inputs. Circuits of exponential size have to be considered as unrealistic ones from the same reasons as exponential time is considered as an intractable one. Therefore the attraction of researchers has focused to an interesting intermediate case — namely that of neural circuits of simultaneously polynomial size and constant depth.

Such circuits can be also very powerful — e.g. in Fig. 4 there is a so-called binary sorting circuit , of linear size, that rearranges the 0's and 1's appearing on its input so that 1's precede the 0's on its output, in a constant parallel time.

4.5. *Threshold circuits.* In the complexity theory a somewhat special case of neural circuits hidden under the disguise of so called *threshold circuits* has been extensively studied.

A threshold circuit (TC) is just a neural circuit with unit weights. In fact, both neural circuits from Fig. 3 and 4 are threshold circuits.

The full characterization of the class of functions that can be computed by TC's of polynomial size and constant depth in terms of some standard complexity classes is not known. Nevertheless, it is known that some very important classes are computed by such circuits. Along these lines the important recent result is that by Reif and Tate [RT], who found the surprising relationship between TC's and so-called *finite field $Z_{P(n)}$ circuits* (FFC's). In the latter circuits each node computes either multiple sums or products of integers modulo a prime $P(n)$. FFC's are especially suitable for the realization of various basic numerical algorithms like simple and iterated addition and multiplication, computing integer reciprocal, etc.

The main result of Reif and Tate is that all functions computed by TC's of size $S(n) \geq n$ and depth $D(n)$ can also be computed by FFC's of size $O(S(n) \, log \, S(n) + nP(n) \, log \, P(n))$ and depth $O(D(n))$, and vice versa, all functions computed by FFC's of size $S(n)$ and depth $D(n)$ can be computed by TC's of size $O((S(n) \, log \, P(n)^{1+\epsilon}/\epsilon^2)$ and depth $O(D(n)/\epsilon^5)$.

They got many useful and quite surprising consequences of this result. For example, integer reciprocal can be computed in size $n^{O(1)}$ and depth $O(1)$. More generally, any analytic function (such a sine, cosine, exponentiation, square root, and logarithm) can be computed within accuracy 2^{-n^c}, for any constant c, by TC's of polynomial size and constant depth. In addition, integer and polynomial quotient and remainder, FFT, polynomial interpolation, Chinese Remaindering, all the elementary symmetric functions, banded matrix inverse, and triangular Toeplitz matrix inverse can be exactly computed by such TC's. For details see the original paper [RT].

5. Symmetric neural networks

5.1. *Basic definition.* Symmetric neural networks are neural networks with an undirected interconnection graph containing no loop-edges:

Definition 5.1. *A symmetric neural network M is a neural network for which $a_{i,j} = a_{j,i}$ and $a_{i,i} = 0$, where $a_{i,j}$ is the weight of an edge connecting an i-th neuron with a j-th neuron in M.*

Symmetric neural networks are often termed *Hopfield neural networks* since it was apparently Hopfield who as the first recognized that these special instances of neural networks are of particular interest. Namely, for any Hopfield network the termination of any computation in the sequential mode can always be guaranteed (see Section 5.3.)! Moreover, there exist some natural physical realizations just of this type of neural networks — viz. Ising spin glasses [B], or certain models of so–called 'optical computers' [FPPP].

It is therefore quite surprising that until recently the computational power of symmetric neural networks has not been known [ESM, F] as it was conjectured that perhaps these networks need not be as powerful as asymmetric ones since the former are but a special case of the latter ones.

5.2. *The computational power of symmetric neural networks.* We shall show now the result by Wiedermann [W1, W2] that the computational power and efficiency of symmetric neural networks is no less as that of neural circuits (the independent proof is given also in [P]). To prove this claim we shall need the following definition.

Definition 5.2. *We shall say that a given neuron u (with symmetric weights) has the insensitivity range $\langle a, b)$, with $a \leq 0$, $b > 0$, if the addition of a further input with weight $w \in \langle a, b)$ will not affect the activity of u (i.e., its behavior will further depend only on the original inputs).* ∎

In the proof of the following lemma we shall see that the definition of insensitivity range is correct, i.e., that the insensitivity range of any neuron always comprises an interval of form $\langle a, b)$, with $a \leq 0$ and $b > 0$. The lemma actually says more:

Lemma 5.1. *For any neuron u and any $\alpha \leq 0$ and $\beta > 0$ there is an equivalent neuron v that computes the same function as u does, and with insensitivity range $\langle \alpha, \beta)$.*

Proof (Sketch). Let $w_1, w_2, ..., w_k$ be the input weights of u and t its threshold. Define $a = max\{\sum_{i=1}^{k} w_i x_i \mid \sum_{i=1}^{k} w_i x_i < t, x_i \in \{0,1\}\}$ and $b = min\{\sum_{i=1}^{k} w_i x_i \mid \sum_{i=1}^{k} w_i x_i \geq t, x_i \in \{0,1\}\}$.

Clearly $a < t \leq b$ and $\langle t - b, t - a)$ is the insensitivity range of u, for any $t \in (a, b\rangle$. Select now such a $t_0 \in (a, b\rangle$ that splits the interval $(a, b\rangle$ in the same ratio in which 0 splits the interval $\langle \alpha, \beta)$ — i.e., $t_0 = (\alpha a - \beta b)/(\alpha - \beta)$. To obtain the weights and the threshold of v multiply all weights of u and t_0 by $(\beta - \alpha)/(b - a)$. ∎

Now we are ready to formulate the main result of this subsection.

Theorem 5.1. *Any neural circuit C of size $S(n)$ and depth $D(n)$ can be simulated by a symmetric neural network N of size $S(n)$ in parallel time $O(D(n))$.*

Proof (Sketch). The main idea in the construction of N is to adjust the weights and the thresholds of each neuron in C with the help of Lemma 5.1 so as the total minimal and maximal sum of its output weights would lie in the insensitivity range of each neuron. This will enable then to introduce to each output connection the symmetric connection with the same weight — i.e., the transformation of C to N. To do so start with the set of neurons of C that have no successors in C and leave their weights and thresholds as they are and consider these neurons as being already adjusted. Now proceed recursively as follows : for each neuron v whose weights have already been adjusted compute the minimal sum α and the maximal sum β of its output weights. Then adjust the input weights and the threshold of v with help of Lemma 5.1 so that the insensitivity range of v would be $\langle \alpha, \beta)$. The process will stop at input neurons that have no predecessors.

As a result we obtain a circuit C' equivalent to C. To obtain N introduce the backward connections to existing ones in C' with the same weights and note that these connections can by no means affect the behavior of the corresponding target neurons since their contribution lies always in the insensitivity range of target neurons.

Thus the neurons that are farther from the input neurons cannot affect those that are closer to them; hence in a sense the computation is directed from input neurons towards the output ones. Therefore the computation time will be $O(D(n))$. ∎

Thus, recalling also the result of Theorem 4.1 it follows that from a computational point of view there is no difference between the computational power of neural circuits and that of symmetric neural networks. The transformation from the previous theorem can be of practical significance e.g. in a case when the technological constraints allow only for realization of Hopfield networks.

5.3. *The termination of sequential symmetric network computations.* Hopfield [H2] has shown that the computation of any symmetric neural network can be thought of as a process of a minimization of a certain energy function. This energy function takes the form $E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} a_{i,j}x_ix_j + \sum_{i=1}^{n} t_ix_i$, with $a_{i,j} = a_{j,i}$, $a_{i,i} = 0$, and the meaning of individual symbols as described in Section 3.2. Hopfield proved in fact the following theorem that makes symmetric neural networks so attractive:

Theorem 5.2. *Starting from any initial configuration any symmetric neural network with energy function E computing in a sequential mode will achieve a stable state after at most $O(p)$ computational cycles, where $p = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} |a_{i,j}| + \sum_{i=1}^{n} |t_i|$; moreover this stable state represents a local minimum of E.*

Proof (Sketch). The change ΔE in E due to changing the state of i–th neuron by Δx_i is $\Delta E = -\sum_{j=1}^{n}[a_{i,j}x_j - t_j]\Delta x_i$. According to the mechanism of neural network computation the change of x_i is positive if and only if the expression in the bracket is positive and similarly for the negative case. Thus any action of any neuron cannot cause the increase of the value of E and whenever some neuron changes its state the value of E will decrease. Since $|E|$ is bounded by p after at most p computational cycles the network must reach a stable state which is a local minimum of E. ■

5.4. *Nondeterministic computations and energy function minimization.* From Theorem 5.2 it follows that the computation of any symmetric neural network in sequential mode will always terminate in some final configuration in which the corresponding energy function achieves its minimum. Which minimum will be achieved — whether a local or a global one — depends, of course, on the initial configuration and in the sequential computational mode also on the order in which neurons sample their inputs during each computational cycle.

Later on we shall see that certain computations of symmetric neural networks are of special interest — namely those for which the corresponding energy function achieves its global minimum at the end of computation. Such computations correspond to successfull nondeterministic computations (see Corollary 5.3.1). Therefore it is worth to study the minimization problem of energy functions of such networks. Unfortunatelly, the following theorem by Wiedermann [W2] shows that the above minimization problem is a difficult one.

Theorem 5.3. *Let N be a symmetric neural network with weights of at most polynomial size in the size of N. Then for any integer k the problem of deciding whether there exists an initial configuration of N for which a stable state with energy not greater than k will be achieved is an NP–complete problem.*

Proof (Sketch). First we shall show that the above problem is in NP. Consider therefore a nondeterministic Turing machine M that simulates N. M first guesses the initial configuration of N. This takes time polynomial in the size of N since the size of each configuration is linear. Then for this configuration M simulates the computation of N. According to Theorem 5.2. this simulation will end in polynomial time due to our assumption concerning the size of weights of N.

The computation of M ends successfully if and only if for our initial configuration a stable state with energy $\leq k$ is achieved.

Thus the total running time of M's simulation is polynomial and hence our problem belongs to NP.

Next we shall construct a special symmetric network N with energy function E that tests the satisfiability of a given boolean formula f in a conjunctive normal form with n variables. It is known that the satisfiability problem of such formulae presents an NP–complete problem [GJ]. Then we will show that there is a constant k such that f is satisfiable if and only if there is an initial configuration for which a local minimum of E with a value $\leq k$ is achieved.

The schema of N is depicted in Fig. 4.

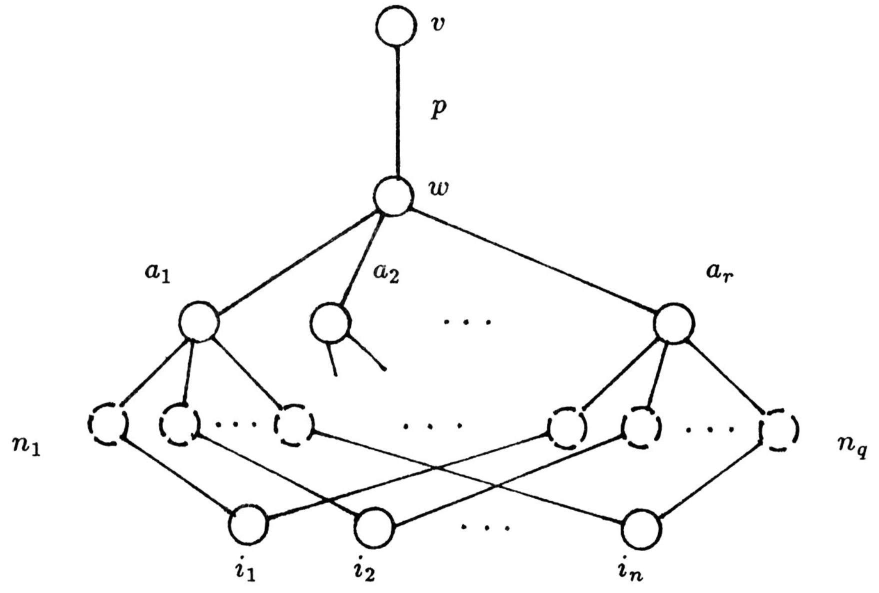

Fig. 4 A schema of a symmetric network for satisfiability testing

In this figure the thresholds of only some neurons that will be important in the following explanation are given in circles representing the corresponding neurons; similarly important edges are labeled by their weights.

The states of neurons $i_1, i_2, ..., i_n$ play the role of boolean variables in f ; neurons $n_1, n_2, ..., n_q$ are negating neurons that compute literals (they are present only when the respective variable has to be negated in the corresponding clause of f). Neurons a_1, a_2,-$..., a_r$ compute multiple OR's — i.e., individual clauses of f and the neuron v computes the multiple AND of all clauses — i.e., the value of f on the input represented by states of $i_1, i_2, ..., i_n$.

The purpose of w is to decrease the value of E as low as we wish in the case that v is active; this is achieved by choosing p large enough. Note that when neurons v and w are both active they contribute with a value of $\Theta(p)$ to the energy function.

In the initial configuration of N the neurons i_j's corresponding to variables in f represent the input neurons . The states of all other neurons are initialized to 0.

Under this arrangement it follows that for some initial configuration the neuron v could be active in some stable state if and only if f is a satisfiable formula.

Consider now the corresponding energy function E. It is clear by now that by a suitable choice of p we can achieve that the value of E is $\leq k$ for any computation that starts in any initial configuration that satisfies f.

Finally note that the value of p need not be greater than the one used in Theorem 5.1. and that all weights in N, and the size of N, is polynomial in the length of f. Therefore the reduction from f to N (and hence to E) takes polynomial time. ∎

Corollary 5.3.1. *Let M be an arbitrary single–tape nondeterministic Turing machine of time complexity $T(n) \geq n \log n$. Then there is a symmetric neural network N of size $O(T(n))$ with energy function E and a constant k such that M accepts its input if and only if there is such an initial configuration of N for which a stable state with energy $\leq k$ is achieved.*

Proof (Sketch). There is a reduction [R] from a single–tape nondeterministic Turing machine with time complexity $T(n) \geq n \log n$ to a boolean formula f in conjunctive normal form of length $T(n)$ which is satisfiable if and only if the machine accepts its input. For this formula use the construction from the previous theorem. ∎

Observe the 'cautious' formulation of the last corollary: it is not claimed here that N will always find a solution of the original problem. No doubt that the network will converge to some stable state but not necessarily to that in which the value of E is $\leq k$. The network will converge to that local minimum that is a so–called *attractor* of the initial configuration of the network. Hence the convergence can be 'directed' by a suitable choice of initial states of certain neurons — but Theorem 5.3. and its proof show that exactly this presents an NP–complete problem by itself! This also seems to be the bottleneck of analog Hopfield networks when used for solving NP–complete problems (see e.g. [HT2]) where in order to obtain a correct or a good approximate solution, it was necessary to set the initial values of analog variables so as they lie in the region of attraction of sufficiently low local minimum of the corresponding energy function.

Note also that if there were a uniform family of symmetric neural networks of polynomial size, with weights also of polynomial size that would always find a solution of some NP–complete problem, it would imply that $P = NP = co - NP$!

5.5 *The terminating problem of parallel symmetric neural networks.* Providing the sequential computational mode Theorem 5.2 guarantees the termination of any symmetric neural network computations. Surprisingly the result does not hold for parallel computational mode as seen from the simple example of a symmetric neural network consisting od two zero–threshold neurons connected by an edge with weight equal to -1. Starting from an initial configuration in which both neurons are in the same state the network will flip–flop for ever between two configurations. However, the computation of the same network will terminate when started in a configuration with different states of both neurons. This observation can be generalized to the case of arbitrary symmetric neural networks [BG,P].

Theorem 5.4. *Starting from any initial configuration after at most $O(p)$ parallel computational steps any symmetric neural network N will alternate between two configurations or will achieve a stable state (where p is the same as in Theorem 5.2).*

Proof (Sketch). We shall construct a symmetric neural network M that will simulate N in a sequential mode. M will consist of two sets of neurons — M_0 and M_1, respectively. There is one-to-one correspondence between neurons of N and M_0, and N and M_1 (and therefore M_0 and M_1 are of equal cardinality). There are no connections among the neurons in M_0 and the same holds for M_1. However, if in N there is a neuron u that is connected with neurons $u_1, \ldots, u_k$ for some $k \geq 1$, then the corresponding neuron in M_0 is connected with the corresponding neurons in M_1, and vice versa. Doing so, the respective weights and thresholds remain the same as in N.

At the beginning of the simulation of N by M the neurons in M_0 are initialized similarly as the neurons in N; neurons in M_1 are set into passive states. Then the sequential simulation can start. First, the states of all neurons in M_1 are updated sequentially, then again the states of all neurons in M_0, etc.

It is clear that during the simulation the following invariant holds: for i even (odd), after updating the states of all neurons in M_0 (M_1), the configuration of neurons in M_0 (M_1) corresponds to the configuration of neurons in N after i–th parallel step of N.

Since our simulation is sequential due to Theorem 5.2 it must terminate in some stable configuration after at most $O(p)$ computational cycles (note that the energy function of M differs from that of N only by the multiplicative factor of 2). When in this stable configuration of M the configuration of neurons in M_0 is different from those in M_1 the original parallel network N will alternate between these two configurations; otherwise it will stop as well. ∎

6. Boltzmann Machines

6.1. *Escaping from local minima.* In the last section we have demonstrated a close connection between nondeterministic computations and the minimization of energy function that corresponds to some symmetric neural network. We have seen that in order to get the correct result of the original nondeterministic computation it was necessary to ensure somehow, by a choice of a suitable initial configuration, the convergence of the computation to some 'sufficiently low' local minimum — of course, the best of all, to a global minimum.

However, since the finding of such a suitable initial configuration presents an NP-complete problem by itself, as we have proved, even without knowing the theoretical reason physicists devised other, in fact heuristics techniques how to increase the probability of arriving into some acceptable local minimum of energy function.

Roughly, the idea is as follows. We start from any initial configuration and wait until the network converges by repeatedly transiting from a given configuration to a configuration with a lower energy, in accordance with Theorem 5.2., into some stable state. Now, if the value of the corresponding energy function is not sufficiently low, we try to 'escape' from the local minimum we are in by changing temporarily the mode of neuron computations! This is achieved by allowing also transitions from configurations to other configurations with higher energy.

6.2. *Simulated annealing.* The mechanism that allows for escaping from local minima of energy functions is the probabilism. Namely, we arrange the things so that the neurons are not necessarily obliged to change their states as dictated by the deterministic rule described in Section 3, i.e., depending on whether the total input exceeds the corresponding threshold value or not. Rather, this change becomes now a subject of a probabilistic decision — the neurons will change their state only with a certain probability that can be controlled, so to speak, from outside.

From historical reasons in analogy with statistical mechanics (the behavior of systems with many degree of freedom in thermal equilibrium at a finite temperature) the process of probability approaching 1 in the behavior of neurons is termed 'temperature lowering' or 'cooling'.

Thus, to escape from a local minimum (which, in physical terms, corresponds to a complete 'freezing' of a system) we start to increase the temperature slowly in order to come temporarily into higher energy states, and than again we start with cooling, in hoping that we arrive into another local minimum, with lower energy than before.

The whole process is appropriately termed *simulated annealing* and its history goes back to Metropolis et al. [MRRTT], and Kirpatrick et al. [KGV]. The corresponding device is called a *Boltzmann machine*, and the above physical motivation explains how Boltzmann's name came into the play.

6.3. *The Boltzmann machine definition.* Formally, a discrete Boltzmann machine is modeled by a symmetric neural network with neurons enhanced by a probabilistic mechanism.

Definition 6.1. *A discrete Boltzmann machine is a symmetric neural network enhanced by a so-called temperature function $\tau : N \times Z \rightarrow Z$ which assigns a 'temperature' to each neuron and time.* ∎

The state of each neuron is updated as follows. The total input to the i-th neuron u_i in time $m > 0$ is given similarly as in Section 4 by the sum $h_i(m) = \sum_{i=0}^{n} a_{i,j}x_j(m-1)$, where $a_{i,j} \in Z$ is the weight of an edge connecting u_i with u_j, and $x_j(m)$ is the state of u_j in time m. Then u_i is active in time m with some probability $p(h_i(m) - t_i, m) = p(\Delta_i(m), m)$, where t_i is the threshold of u_i. Typically, in the literature the activation probability function $p(\Delta_i(m), m) = 1/(1 + e^{-\Delta_i(m)/\tau(i,m)})$ is recommended.

Note that as τ approaches ∞, the above probability approaches $1/2$, and thus the network behaves more or less unpredictably, and as τ approaches 0, the neurons start to act deterministically, as in an ordinary neural network.

The machine usually operates in a parallel mode.

6.4. *The art of simulated annealing.* There is a lot of interesting and successful experiments with simulated annealing used in solving various problems (like traveling salesman problem and wire routing [KGV], independent set, max cut, graph coloring [KA]) described in the literature. Most of the results were obtained with the help of an *analog Boltzmann machine*, i.e., such where the 'state' of neurons can take continuous values between 0 and 1.

Computational experiments with these machines seem to indicate that the convergence to sufficiently minimal energy states need not occur if the temperature is lowered too fast through many different levels. Thus various annealing schedules that could guarantee the convergence to the global optimum have been investigated. Numerical experience of most researchers recommends the following procedures to yield the best results: iterate at a fixed temperature 'long enough' before lowering the temperature to the next level. Then usually few different temperatures were sufficient.

Theoretical explanation why the latter procedure should be preferred over the first one is given in [FS]. However, it must be noted that in general the convergence to a global minimum could not be guaranteed and that the above methods can give acceptable results only when good, not necessarily optimal solutions are sufficient to obtain.

6.5. *Boltzmann machines and neural circuits.* Bellow we shall sketch a recent and a quite surprising result by Parberry and Schnitger [PS1, PS2] that Boltzmann machines are not much more powerful than neural circuits introduced in Section 4.

More precisely these authors have shown that any Boltzmann machine of polynomial time complexity as defined in Section 6.3 can be simulated by a neural circuit with running time greater by a constant factor and size greater by a polynomial. It means that probabilism and the related ability to perform simulated annealing are unimportant from the complexity point of view!

The proofs of the above claim are based on known techniques from theory of iterative arrays and combinatorial circuits. First, the cycles are removed from the interconnection graph of a Boltzmann machine B at hand using the method of 'unwrapping' the computation in time, i.e., a circuit is build in which i–th layer corresponds to a configuration of B immediately after performing its i–th computational cycle (similarly as in the proof of Theorem 4.1). Then, the probabilism is removed from neurons firstly by replacing each neuron by a small number of deterministic neurons with random inputs, and then by using the well-known technique by Adlemann [A] that transforms the above circuit with random inputs to a completely deterministic one. As a result a neural circuit equivalent to B with complexity characteristics as above is obtained.

6.6. *The computational limits of Boltzmann machines.* Although occasionally some experimental researchers report that Boltzmann machines are surprisingly good in solving some NP–complete problems where only exact rather than approximate solutions make sense (like in the 3-satisfiability problem — see e.g. [J]), and thus seem to indicate that it might be the case that $P = NP$, we will further give a strong evidence why such claims should be regarded with the greatest possible care.

Our reasoning will be based on some of our earlier results mentioned here and, for the sake of correctness, it must be stressed that it holds only for discrete Boltzmann machines.

Imagine therefore that we wish to solve some NP–complete problem P with the help of a Boltzmann machine B as described in Section 6.2. Due to the last result from Section 6.5 there is a neural circuit C of polynomial size that is equivalent to B. In turn, according to Theorem 4.1, C is further equivalent to a symmetric neural circuit N. To solve P on N we have to initialize N so as its computation ends in a stable state with sufficiently low energy. According to Theorem 5.3 this presents an NP–complete problem by itself!

Thus we see that unless $P = NP$ even discrete Boltzmann machines do not present a way to go round the intractability of NP–complete problems.

7. Conclusions

7.1. *Summary.* In the previous review we have presented basic models of abstract discrete neural devices: single neurons, neural circuits, symmetric neural networks, and

Boltzmann machines. We have seen that except single neurons, whose computational power is restricted to linearly separable boolean functions, all the other devices are equivalent in the sense that they can compute any recursive function. For a given boolean function we have shown that the problem of deciding whether there exists an equivalent neural representation presents an intractable problem. Moreover, we have seen that all the above neural networks can simulate each other with at most polynomial increase in size and a constant increase in time — whether in a parallel or in a sequential computational mode.

The link to traditional models of computations was provided via combinatorial circuits. It is well-known that combinatorial circuits with bounded fan-in belong to a so-called *second machine class* [vEB] that embodies machines allowing for unrestricted parallelism. For instance, this class includes alternating Turing machines, vector machines, array processing machines, various versions of parallel RAM's, etc. It follows that neural circuits with bounded fan-in belong also to the second machine class, while these with unbounded fan-in are outside of this class, since they can compute any boolean function in a constant depth.

For symmetric neural networks it was shown that there is a close connection between nondeterministic computations and minimization of corresponding energy functions.

Finally, the evidence was presented that even discrete Boltzmann machines, with their added ability to perform simulated annealing, are not much more powerful than simple models of neural nets.

7.2. *The significance of neurocomputing* . From the complexity point of view the previous results demonstrate that models of discrete neurocomputing just enrich the classic repertoire of computational models by devices that are inspired by biological or physical motivations — depending on which framework is preferred. The computational power and efficiency of these models have been studied already for years and our review shows in fact that most of results and open problems achieved in neural formalism can be translated into other formalisms, and vice versa. Nevertheless, there are also results specific to neurocomputing — most notably those concerning the relation between nondeterministic computations and energy function minimization – that have brought new insights into general mechanisms of computations.

Last but not least, the significance of discrete neurocomputing should be seen also on a methodological level, where these models provide new natural conceptual tools for modeling some problems that we believe can be related to brain activities.

References

[A] Adlemann, L.: Two Theorems on Random Polynomial Time, *Proc. 19-th FOCS*, Washington D. C., 1978, pp. 75–83

[AH] Ackley, D. N. — Hinton, G. E. — Sejnowski, T. I.: A Learning Algorithm for Boltzmann Machines. *Cognitive Science* **9**, 1985, pp. 147–169

[B] Barahona, F.: On the Computational Complexity of Ising Spin Glass Models. *J. Phys. A.* **15**, 1982, pp. 3241–3253

[BG] Bruck, J. — Goodman, J. W.: A Generalized Convergence Theorem for Neural Networks and Its Application in Combinatorial Optimization. *Proc. IEEE First International Conf. on Neural Networks*, Vol. **3**, 1987, pp.649–656

[CKS] Chandra, A. K. — Kozen, D. C. — Stockmeyer, L. I.: Alternation. *JACM* **28**, 1981, pp. 114–133

[CSV] Chandra, A. K. — Stockmeyer, L. I. — Vishkin, U.: Constant Depth Reducibility. *SIAM J. Comput. Vol.* **15, No. 3**, 1984, pp. 423–432

[ESM] Egecioglu, O. — Smith, T. R. — Moody, I.: Computable Functions and Complexity in neural networks. *Tech. Rep. ITP–124*, University of California, Santa Barbara, 1986

[FPPP] Farhat, N. H. — Psaltis, D. — Prata, A. — Paek, E.: Optical Implementation of the Hopfield Model. *Applied Optics*, **24**, 1985, pp. 1469–1475

[FS] Faigle, U. — Schrader, R.: On the Convergence Of Stationary Distributions in Simulated Annealing Algorithms. *Inf. Proc. Letters*, **27**, 1988, pp. 189–194

[F] Feldman, J. A.: Energy and the Behavior of Connectionist Models. *Tech. Rep. TR–155*, University of Rochester, Nov. 1985

[GJ] Garey, M. R. — Johnson, D. S.: Computers and Intractability. A Guide to the Theory of NP–Completeness. *Freeman and Co.*, San Francisco, 1979

[H1] Hopfield, J. J.: Neural Networks and Physical Systems with Emergent Collective Computational Abilities. *Proc. Natl. Acad. Sci. USA* **79**, 1982, pp. 2554–2558

[H2] Hopfield, J. J.: Neurons with Graded Response Have Collective Computational Properties Like Those of Two–state Neurons. *Proc. Natl. Acad. Sci. USA*, 1984, pp. 3088–3092

[H3] Hopfield, J. J.: The Effectiveness of Neural Computing. *Proc. IFIP'89*, North-Holland, l989, pp. 503–507

[HT1] Hopfield, J. J. — Tank, D. W.: 'Neural' Computations of Decisions in Optimization Problems. *Biol. Cybern.* **52**, 1985, pp. 141–152

[HT2] Hopfield, J. J. — Tank, D. W.: Computing with Neural Circuits: A Model. *Science* **233**, 1986, pp.625–633

[J] Johnson, J. l.: A Neural Network Approach to the 3-Satisfiability Problem. *J. of Parall. and Distrib. Comput.* **6**, 1989, pp. 435–449

[KGV] Kirkpatrick, S. — Gellat, C. D., Jr. — Vecchi, M. P.: Optimization by Simulated Annealing. *Science*, **220**, No. 4598, 1983

[KA] Korst,. J. H. M. — Aarts, E. H. L.: Combinatorial Optimization on a Boltzmann Machine. *J. of Parall. and Distrib. Comput.* **6**, 1989, pp. 331–357

[MRRTT] Metropolis, N. — Rosenbluth, A. — Rosenbluth, M. — Teller, A. — Teller, E.: *J. Chem. Phys.*, **21**, 1087, 1953

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

[Mu] Muroga, S.: Threshold Logic and Its Applications. *Wiley–Interscience*, New York, 1971

[MP] Minsky, M.— Papert, S.: Perceptrons. An Introduction to Computational Geometry. *The MIT Press*, Cambridge, Mass., 1969

[MTB] Muroga, S. — Tsubi, T. — Baugh, Ch. R.: Enumeration of Threshold Functions of Eight Variables. *IEEE Trans. on Comp.*, C-19, No. 9, 1970, pp. 818–825

[P] Parberry, I.: A Primer on the Complexity Theory of Neural Networks. *Research Report CS-88-38*, Dept. of Comp. Sci., The Pennsylvania state university, October 1988

[PS1] Parberry, I. — Schnitger, G.: Parallel Computation with Threshold Functions. *JCSS* **36**, 1988, pp. 278–302

[PS2] Parberry, I. — Schnitger, G.: Relating Boltzmann Machines to Conventional models of Computations. *Neural Networks*, **2**, 1989

[RT] Reif, J. H. — Tate, S. R.: On Threshold Circuits and Polynomial Computation. *Technical Report*, Dept. of Comp. Sci., Duke University, 1988

[R] Robson, J. M.: Linear Size Formulas for Non–deterministic Single Tape Computations. *Proc. 11–th Australian Comp. Sci. Conference*, Brisbane, Feb. 3–5, 1988

[vEB] van Emde Boas, P.: Machine Models and Simulations. *ITLI Prepublication Series of Computation and Complexity Theory CT–88–95*, University of Amsterdam, 1988

[W1] Wiedermann, J.: On the Computational Power of Neural Networks and Related Computational Systems. *Technical Report OPS-9/1988*, Department of Programming Systems, VUSEI-AR, Bratislava, June 1988 (in Slovak), also in *Proc. SOFSEM'88*, VUSEI-AR Bratislava, November 1988, pp. 73–78

[W2] Wiedermann, J.: On the Computational Efficiency of Symmetric Neural Networks. *Proc. 14-th Symp. on Math. Found. of Comp. Sci., MFCS'89*, LNCS Vol. **379**, Springer Verlag, Berlin, 1989, pp. 545–552