The Computational Theory of Neural Networks

Šíma, Jiří
2000

INSTITUTE OF COMPUTER SCIENCE

ACADEMY OF SCIENCES OF THE CZECH REPUBLIC

The Computational Theory of Neural Networks

Jiří Šíma

Technical report No. 823

December 2000

Institute of Computer Science, Academy of Sciences of the Czech Republic
Pod vodárenskou věží 2, 182 07 Prague 8, Czech Republic
phone: (+420 2) 6605 3030 fax: (+420 2) 85 85 789
e-mail: sima@cs.cas.cz

INSTITUTE OF COMPUTER SCIENCE

ACADEMY OF SCIENCES OF THE CZECH REPUBLIC

The Computational Theory of Neural Networks

Jiří Šíma [1]

Technical report No. 823
December 2000

Abstract

In the present paper a detailed taxonomy of neural network models with various restrictions is presented with respect to their computational properties. The criteria of classification include e.g. feedforward and recurrent architectures, discrete and continuous time, binary and analog states, symmetric and asymmetric weights, finite size and infinite families of networks, deterministic and probabilistic models, etc. The underlying results concerning the computational power of perceptron, RBF, winner-take-all, and spiking neural networks are briefly surveyed and completed by relevant references.

Keywords

neural networks, perceptrons, RBF networks, winner-take-all networks, spiking neurons, computational power

[1]Research supported by grants GA ČR No. 201/98/0717, GA AS CR No. B2030007, and by a personal grant from the University of Jyväskylä.

1 Introduction

The (artificial) neural networks represent a widely applied computational paradigm that is an alternative to the conventional computers in many areas of artificial intelligence. By analogy with classical models of computation such as Turing machines which are useful for understanding the computational potential and limits of conventional computers, the capability of neural networks to realize general computations have been studied for more than decade and many relevant results have been achieved [25, 44, 84, 88, 92, 93, 94, 103, 108, 123, 131]. In particular, the computational and descriptive power of neural nets have been investigated by comparing their various architectures with each other and with more traditional computational models and descriptive tools including finite automata, regular expressions, grammars, Turing machines, Boolean circuits, etc. The aim of this approach is to find out what is, in principle, or efficiently, computable by particular neural models, and how to optimally implement required functions. Thus, the neural networks are classified into a computational taxonomy which may enrich the traditional repertoire of computational means and even bring more efficiency. In addition, neural networks have been inspired by neurophysiological knowledge and hence, the respective theoretical results may help to broaden our understanding of the computational principles of mental processes.

First recall a general model of a neural network which consists of s simple computational *units* or *neurons*, indexed as $V = \{1, \ldots, s\}$ where $s = |V|$ is called the network *size*. Some of these units may serve as external inputs or outputs and hence, assume the network has n *input* and m *output* neurons, respectively. The remaining ones are called *hidden* neurons. The units are densely connected into an oriented graph called *architecture* in which each edge (i, j) leading from neuron i to j is labeled with real *(synaptic) weight* $w(i, j) = w_{ji} \in \Re$. The absence of a connection within the architecture corresponds to a zero weight between the respective neurons.

The *computational dynamics* of a neural network determines for each neuron $j \in V$ the evolution of its real *state* (*output*) $y_j^{(t)} \in \Re$ in time $t \geq 0$. This establishes the *network state* $\mathbf{y}^{(t)} = (y_1^{(t)}, \ldots, y_s^{(t)}) \in \Re^s$ at time $t \geq 0$. At the beginning of computation, the neural network is placed in an *initial state* $\mathbf{y}^{(0)}$ which may also include an external input. Typically, a network state is updated as certain neurons collect their *inputs* from the outputs of incident neurons via the underlying weighted connections and transform them to their current states. Finally, an output from the network is read at the end or even in the course of computation. The *digital* computations over external *binary* inputs providing *binary* outputs will mainly be assumed here although their values may be encoded by analog states and the intermediate stage of computation often operates with real numbers.

The neural networks can be classified according to the restrictions that are imposed on their parameters and/or computational dynamics. In this way various models are obtained which have different computational capabilities. In the present paper a detailed taxonomy of neural network models is presented from the computational point of view. The respective theoretical results concerning the computational power of neural networks are briefly surveyed and completed by relevant references.

The main focus is on the classical *perceptron networks* in Section 2 whose computational properties are now well understood. Both their *discrete-* (Section 2.1) and *continuous-time* (Section 2.2) dynamics are considered. Besides a single perceptron (Paragraph 2.1.1), the models of discrete-time perceptron networks are basically divided according to their architectures into *feedforward* (Paragraph 2.1.2) and *recurrent* (Paragraph 2.1.3) networks corresponding to finite and possibly infinite computations, respectively. This architectural criterion is also applied in the classification of their *probabilistic* (Paragraph 2.1.4) versions in Paragraphs 2.1.4.1 and 2.1.4.2. The computational taxonomy of the discrete-time perceptron networks is further refined in the *binary* (e.g. Paragraph 2.1.2.1) and *analog* networks (e.g. Paragraph 2.1.2.2) regarding the different domains of their state values. Also the *finite* networks (e.g. Paragraph 2.1.3.1) and their *infinite* families (e.g. Paragraph 2.1.3.2) are distinguished with respect to the input protocol which can be online or off-line, respectively. In addition, the computational properties of the *symmetric* recurrent networks (Paragraph 2.1.3.1.2) with undirected architectures differ from that of the *asymmetric* ones (Paragraph 2.1.3.1.1).

Finally, in Section 3 other neural network models whose computational capabilities have only recently been studied are surveyed. This includes the *RBF* (Section 3.1) and *winner-take-all* (Section 3.2) networks which employ computational units that are alternative to the perceptrons. In addition, the computational taxonomy is extended with the networks of biologically more plausible *spiking neurons* (Section 3.3) that make use of temporal coding of their state values. Both the *deterministic* (Paragraph 3.3.1) and *noisy* (Paragraph 3.3.2) spiking networks are reviewed.

2 Perceptron Networks

The *perceptron* or *threshold* networks [83, 102] represent the most popular neural network model. The corresponding computational dynamics will be inspected for discrete and continuous time.

2.1 Discrete-Time Dynamics

The perceptron network, working in discrete time, updates its state only at time instants $t = 1, 2, \ldots$. An *excitation*

$$\xi_j^{(t)} = \sum_{i=0}^{s} w_{ji} y_i^{(t)} \tag{2.1}$$

is assigned to each neuron j at time $t \geq 0$ as the respective weighted sum of inputs including a *bias* $w_{j0} \in \Re$ which can be viewed as the weight of the formal constant unit input $y_0^{(t)} = 1$, $t \geq 0$. At the next instant $t+1$, only the neurons $j \in \alpha_{t+1}$ from a certain subset $\alpha_{t+1} \subseteq V$ compute their new outputs $y_j^{(t+1)}$ by applying an *activation function* $\sigma : \Re \longrightarrow \Re$ to $\xi_j^{(t)}$ as follows:

$$y_j^{(t+1)} = \sigma\left(\xi_j^{(t)}\right) , \quad j \in \alpha_{t+1} \tag{2.2}$$

while the remaining units $j \notin \alpha_{t+1}$ do not change their states, that is $y_j^{(t+1)} = y_j^{(t)}$ for $j \notin \alpha_{t+1}$. In this way the new network state $\mathbf{y}^{(t+1)}$ at time $t+1$ is determined. The perceptron networks with *binary states* $y_j \in \{0,1\}$ usually employ the *hard limiter* or *threshold* activation function

$$\sigma(\xi) = \begin{cases} 1 & \text{for } \xi \geq 0 \\ 0 & \text{for } \xi < 0\,. \end{cases} \tag{2.3}$$

Sometimes when more appropriate, bipolar values $\{-1,1\}$ (or even more general discrete domains) can be substituted for binary ones $\{0,1\}$ without any substantial change in the size of weights [93, 94]. The *analog-state* networks, on the other hand, approximate (2.3) with some continuous sigmoid activation function, e.g. the *saturated-linear* function

$$\sigma(\xi) = \begin{cases} 1 & \text{for } \xi \geq 1 \\ \xi & \text{for } 0 < \xi < 1 \\ 0 & \text{for } \xi \leq 0 \end{cases} \tag{2.4}$$

or the *standard* (*logistic*) *sigmoid*

$$\sigma(\xi) = \frac{1}{1+e^{-\xi}}\,. \tag{2.5}$$

Hence the states of analog neurons are generally real numbers, e.g. $y_j \in [0,1]$. For the purpose of computing Boolean functions the analog states y_j of output neurons j can be interpreted as binary outputs 0 or 1 with *separation* $\varepsilon > 0$ if $y_j \leq h - \varepsilon$ or $y_j \geq h + \varepsilon$, respectively, for some fixed real threshold value $h \in \Re$.

2.1.1 Single Perceptron

Clearly, a single perceptron with n external binary inputs computes an n-variable Boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$. It has been known for a long time that not all the Boolean functions can be implemented by a single perceptron, among which the parity (XOR) function represents the most prominent example [83]. Thus, the Boolean functions that are computable by perceptrons which are also called *linearly separable* or *linear threshold functions* create a proper subclass of Boolean functions. This class, on the other hand, includes basic logical functions such as AND, OR, NOT, etc., and is closed e.g. under the negation of both the input variables and/or the output value. The number of linearly separable Boolean functions over n variables has been estimated as $2^{\Theta(n^2)}$ (more precisely the lower and upper bounds are $2^{n^2-O(n)}$ [132] and 2^{n^2}, $n \geq 2$ [18], respectively) which is a very small fraction of all the 2^{2^n} Boolean functions. Moreover, the issue of deciding whether a given Boolean function in a disjunctive or conjunctive normal form is linearly separable, is a co-NP-complete problem [46].

Also the descriptive complexity of linear threshold functions has been studied. It was shown that any linearly separable Boolean function can be implemented by using only *integer* weights [83]. The number of bits that are necessary [45] and sufficient [85] for representing a single integer weight parameter in n-input perceptrons is $\Theta(n \log n)$.

2.1.2 Feedforward Networks

The architecture of *feedforward* (*multi-layered*) neural networks is an acyclic graph. Hence, units in a feedforward network can be grouped into a sequence of $d+1$ pairwise disjoint *layers* $\alpha_0, \ldots, \alpha_d \subseteq V$ so that neurons in any layer α_t are connected only to neurons in subsequent layers α_u, $u > t$. Usually, the first so-called *input layer* α_0 consists of n external inputs and is not counted in the number of layers (e.g. two-layred networks have depth 2). The last so-called *output layer* α_d is composed of m output neurons. The intermediate so-called *hidden layers* $\alpha_1, \ldots, \alpha_{d-1}$ include hidden neurons. The number of layers d excluding the input one is called the network *depth*. Obviously, the computation proceeds from the input layer via hidden layers up to the output layer. At the beginning the states of the input units are set to an external input. In a general step, suppose all the outputs of neurons have been determined up to a certain layer $t < d$, then the states of neurons in the next layer α_{t+1} are computed according to (2.1), (2.2). At the end the states of output neurons α_d represent the result of the computation. In this way the so-called *network function* $\mathbf{f} : \{0,1\}^n \longrightarrow \{0,1\}^m$ is evaluated in parallel time which equals depth d.

The above-described model of feedforward neural networks coincides with (*threshold*) *circuits* (where neurons are called *gates*) whose computational properties have intensively been studied in Boolean circuit complexity [129]. A single circuit C computes a vector (multi-output) Boolean function $\mathbf{f}$ of a fixed number n of variables. For universal computation purposes where inputs are of arbitrary lengths, *infinite families* $\{C_n\}$ of circuits, each C_n for one input length $n \geq 0$, are considered. Within this framework the complexity measures such as size $S(n)$ and depth $D(n)$ are expressed in terms of input length n. Another input protocol assumes that each input Boolean variable is presented together with its negation. In this case any threshold circuit can be modified by increasing its size by at most twice so that it contains only *positive* weights while the depth and size of weights are preserved [41].

2.1.2.1 Binary-State Feedforward Networks In this paragraph the computational capabilities of binary-state feedforward neural networks that employ the threshold activation function (2.3) will be inspected. The simplest case of this model—a single perceptron has already been considered in Paragraph 2.1.1. Hence, it is now clear that threshold gates must be connected into circuits to be able to implement general Boolean functions. The size of such **universal threshold circuit** has optimally been determined as

$$\Theta\left(\sqrt{\frac{m2^n}{n - \log m}}\right) \tag{2.6}$$

which was proved to be sufficient to implement any vector Boolean function $\mathbf{f} : \{0,1\}^n \longrightarrow \{0,1\}^m$ within depth 4 [66]. On the other hand, there are functions that require this number of gates even for unbounded depth [52]. The respective lower [86] and upper [65] bounds had actually been earlier estimated for single-output networks and coincide with (2.6) for $m = 1$. In addition, the corresponding lower bound on the number of connections in threshold circuits computing general Boolean functions which squares (2.6) was derived in [19]. These results imply that the infinite fami-

lies of constant-depth threshold circuits with exponentially many gates are capable to compute any input-output mapping in constant parallel time.

In the Boolean complexity theory additional restrictions are imposed on circuits. For example, in threshold circuits the integer weight parameters are usually bounded by a polynomial $O(n^c)$ (for some constant c) in terms of the input length n which translates to $O(\log n)$ bits for a single-weight representation. Clearly, **polynomial weights** reduce the computational power of threshold circuits since from Paragraph 2.1.1 exponential weights are generally required for the general-purpose perceptrons. However, it is known that unbounded weights in threshold circuits can be replaced with polynomial ones by increasing the depth by at most one layer while only a polynomial overhead in the network size is needed [32]. The explicit construction of the underlying polynomial-weight circuit whose polynomial size-increase is independent of the depth can be found in [33]. Thus, polynomial weights can be assumed in multi-layered perceptron networks when one more parallel computational step is granted.

Another restriction concerns the maximum *fan-in* of gates in the circuit, i.e. the maximum number of inputs to a single unit. The circuits with **bounded fan-in** represent a model of conventional circuit technology in which the number of input wires to each gate is restricted. The unbounded fan-in is a typical property of densely interconnected neural networks which may gain more efficiency. For example, a classical circuit of size s, depth d, and with the maximum fan-in 2 can be implemented by a threshold circuit with unbounded fan-in and $O(s^2)$ gates whose depth is $O(d/\log\log s)$ [16]. This yields $O(\log\log n)$ factor speed-up in polynomial-size feedforward neural networks of unbounded fan-in.

The exponential size (2.6) in the above-discussed universal networks is far from being practically realizable for larger input lengths. On the other hand, there is no specific function known so far that would require superlinear number of gates in circuits of unbounded depth [129]. In fact, many functions can be computed by threshold circuits of **polynomial size and constant depth** as it will be seen below. Apart from implementation reasons this restriction also corresponds to neurophysiological knowledge that suggests a few layers in the human brain structure. Within this context the well-known complexity classes TC^0_d, $d \geq 1$ (also denoted by LT_d) have been introduced which denote all the functions computable by polynomial-size and polynomial-weight threshold circuits of depth d and define the complexity class $TC^0 = \bigcup_{d\geq 1} TC^0_d$ corresponding to the underlying constant parallel-time computations. Obviously, $TC^0_1 \subseteq TC^0_2 \subseteq TC^0_3 \subseteq \cdots$ is a possible TC^0 *hierarchy*. For example, TC^0_1 contains all the functions computable by single perceptrons with polynomial weights which is certainly a proper subclass of TC^0_2, that is $TC^0_1 \subsetneqq TC^0_2$ since from Paragraph 2.1.1 there are functions not computable by single perceptrons. A less known result is the deeper separation $TC^0_2 \subsetneqq TC^0_3$ [41] which is of a great importance for neurocomputing. The witnessing example of a function that belongs to $TC^0_3 \setminus TC^0_2$ is the *Boolean inner product* $IP : \{0,1\}^{2k} \longrightarrow \{0,1\}$ ($k \geq 1$) defined as $IP(x_1, \ldots, x_k, x'_1, \ldots, x'_k) = \bigoplus_{i=1}^{k} AND(x_i, x'_i)$ where $\bigoplus$ stands for the XOR (parity) function that determines whether the number of 1s in its k-bit input is odd. This means that polynomial-size and polynomial-weight three-layered perceptron networks are computationally more powerful than two-layered ones. Unfortunately, the sepa-

ration of TC^0 hierarchy above depth 3 is unknown which represents one of the most important open problems in the complexity theory. It is even possible that NP-complete problems can be computed by depth-three threshold circuits with a linear number of gates. Thus, it is conceivable that TC^0 hierarchy collapses at depth 3, i.e. $TC^0 \subseteq TC^0_3$.

Another important case are **symmetric Boolean functions** whose values are independent of the order of input variables, i.e. their function value depends only on the number of 1s within the input. A typical example of a symmetric function is the parity function. It has been shown that any symmetric function over n variables can be implemented by polynomial-weight threshold circuit of size $O(\sqrt{n})$ and depth 3 [120]. In addition, this size is known to be almost optimal, i.e. it cannot generally be reduced under $\Omega(\sqrt{n/\log n})$ even if any depth and unbounded weights are allowed which follows from lower bound (2.6) (for depth 2 even a matching lower bound $\Omega(\sqrt{n})$ can be achieved from [87]). This also illustrates the fact that the threshold circuits are computationally more powerful than the so-called *AND-OR circuits* that employ only basic logical gates such as AND, OR, NOT. In particular, it was proved that the parity function is not computable by polynomial-size constant-depth AND-OR circuits [28] corresponding to the complexity class AC^0. This led to a conjecture of a weaker TC^0 hierarchy collapse that $AC^0 \subseteq TC^0_3$. A partial result along this line that AC^0 functions are computable by threshold circuits of depth 3 and subexponential size $n^{\log^c n}$ (for some constant c) can be found in [2].

The computational power of polynomial-size and polynomial-weight threshold circuits of small constant depths can further be illustrated by their capability of computing basic **arithmetic functions** [100, 103, 123]. For example, the two-layered perceptron networks of polynomial size and weights can be constructed for comparing two n-bit binary numbers [4], or for computing the sum of two [4] or even multiple sum of n such numbers (from [119] by [33]). Further, the product and division of two n-bit binary numbers, powering to n [119], and sorting [118] of n such numbers can be achieved with three-layered feedforward networks whereas the depth in these circuits (except for powering where this issue is still unresolved) cannot be reduced when polynomial size and weights are required (for product and division this follows from [41]; the proofs for division and sorting can be found in [48, 130] and [118], respectively). Moreover, the multiple product of n n-bit binary numbers is performable in depth 4 [119]. It follows that any analytic function can be implemented with a large precision by perceptron networks of polynomial size and weights within only a small constant number of layers [101].

There are also **trade-off** results known among different complexity measures including size, depth, and connectivity in threshold circuits. For example, one can achieve smaller polynomial-size feedforward networks in some cases of the arithmetic circuits above if the optimal depth is increased by few layers (e.g. see [47] for a two-number multiplier implementation with $O(n^2)$ threshold gates in depth 4). Further, a more general trade-off results have been shown such as for every $d > 0$, the n-variable parity function can be computed by depth-$O(d \log n/\log f)$ threshold circuits of size $O(dn/f^{1-1/d})$ with $O(dnf^{1/d})$ polynomial-weight edges and fan-in bounded by f [121]. The corresponding lower bound on the size of any depth-$(d+1)$, polynomial-weight threshold circuit computing the parity function is $\Omega(n^{1/d}/\log^2 n)$ [121]. Similarly, any n-variable

symmetric function is computable by a depth-$O(d \log n / \log f)$ threshold circuit of edge complexity $O(2^{-d/2} n f^{1/(2^d-1)} \sqrt{\log f})$, or $O(n \log n)$ if $f^{1/(2^d-1)} \sqrt{\log f} = o(\log n)$, and with fan-in bounded by f, for every fixed integer $d > 0$ [124].

Recently, a new complexity measure—so called **total wire length** [62] has been introduced for feedforward networks which plays an important role in VLSI circuit implementations (e.g. within the context of efficient sensory processing where the number of parallel inputs is quite large). One possible model assumes that the gates are placed on different intersection points of a 2-dimensional grid (with unit distance between adjacent intersection points). These units can arbitrarily be connected by wires in the plane which may cross. The minimal value of the sum of wire lengths defines the *total wire length* of the circuit. The k-input (threshold) gates are here considered as microcircuits with unit evaluation time occupying each a set of k intersection points of the grid that are all connected by an undirected wire in some arbitrary fashion. The architectures of feedforward networks that are efficient (e.g. almost linear) in terms of the total wire length must substantially differ from commonly designed circuits since already complete connectivity between two linear-size 2-dimensional layers requires a total wire length of $\Omega(n^{2.5})$ [62]. For example, a Boolean function $P^k_{LR} : \{0,1\}^{2k} \longrightarrow \{0,1\}$ $(k \geq 2)$ defined as $P^k_{LR}(x_1, \ldots, x_k, x'_1, \ldots, x'_k) = 1$ iff there exists $1 \leq i < j \leq k$ such that $x_i = x'_j = 1$ which represents a simple global pattern detection prototype, can be computed by a two-layered network with $2 \log k + 1$ threshold gates and total wire length $O(k \log k)$ [62].

2.1.2.2 Analog-State Feedforward Networks The computational capabilities of binary-state feedforward neural networks will now be compared to that with analog units employing e.g. the standard sigmoid activation function (2.5) (note that the following results are actually valid for much more general classes of activation functions). This is probably the most common computational model used in practical neurocomputing, e.g. in back-propagation learning [104]. Obviously, Boolean threshold circuits can be implemented by analog feedforward networks of the same architecture by increasing the size of weights when a more precise approximation of function (2.3) by (2.5) is needed [79]. On the other hand, it appears that the real state values may bring more efficiency in small analog networks of **constant size**. For example, Boolean functions $F_k : \{0,1\}^{2k} \longrightarrow \{0,1\}$ $(k \geq 1)$ defined as $F_k(x_1, \ldots, x_k, x'_1, \ldots, x'_k) = Majority(x_1, \ldots, k_k) \oplus Majority(x'_1, \ldots, x'_k)$ where $Majority(x_1, \ldots, x_k) = 1$ iff $\sum_{i=1}^k x_i \geq k/2$, and $\oplus$ stands for XOR function, can be computed by two-layered networks having only five analog units and constant weights, with separation $\varepsilon = \Omega(k^{-2})$ (or $\varepsilon = \Omega(1)$ when polynomial weights are allowed). However, F_k cannot be implemented by any depth-2 threshold circuit with constant number of binary-output gates and unbounded weights [79] which confirms that analog feedforward networks are slightly more powerful computational devices than their binary counterparts. A similar but *depth-independent* result was shown for so called *unary squaring* functions $SQ_k : \{0,1\}^{k^2+k} \longrightarrow \{0,1\}$ $(k \geq 1)$ which give $SQ_k(x_1, \ldots, x_k, z_1, \ldots, z_{k^2}) = 1$ iff $(\sum_{i=1}^k x_i)^2 \geq \sum_{i=1}^{k^2} z_i$ and can be computed by only two analog units having polynomial weights, with separation $\varepsilon = \Omega(1)$ while any Boolean threshold circuit that computes SQ_k requires size $\Omega(\log k)$ gates even for unbounded depth and weights [21].

Thus, the size of feedforward networks can sometimes be reduced by logarithmic factor when binary units are replaced by analog ones.

Furthermore, the complexity classes TC^0_d can be generalized to their analog versions $TC^0_d(\sigma)$ $(d \geq 1)$ containing all the functions computable by **polynomial-size** and polynomial-weight, analog-state feedforward neural networks with d layers and separation $\varepsilon = \Omega(1)$ employing activation function (2.5). It turns out that these classes coincide, i.e. $TC^0_d(\sigma) = TC^0_d$ for every $d \geq 1$ [79]. Hence, there is no substantial difference in the computational power of analog- and binary-state feedforward networks of the same depth within polynomial size. This computational equivalence is still valid within polynomial size even for unbounded depth and exponential weights provided that the depth of the (simulating) binary threshold circuits is allowed to increase by a constant factor [20]. Moreover, the functions computable with arbitrary small separation ε by analog feedforward networks of constant depth and polynomial size, having *arbitrary* real weights and employing the *saturated-linear* activation function (2.4) have been shown to belong to TC^0 [70]. Finally, the trade-off lower bound concerning the n-variable parity function have also been generalized for analog feedforward networks with polynomial weights, $d+1$ layers and separation $\varepsilon = \Omega(n^{-c})$ (for some constant $c > 0$), to the size of $\Omega(dn^{1/d-\delta})$ neurons for any fixed $\delta > 0$ [122], which is almost optimal since as we already know $O(dn^{1/d})$ units are sufficient.

2.1.3 Recurrent Neural Networks

The architecture of *recurrent* (*cyclic*) neural networks is, in general, a cyclic graph, and hence, the corresponding computations on these devices may not terminate in finitely many steps. Special attention has also been paid to *symmetric* or *Hopfield* networks whose weights satisfy $w(i,j) = w(j,i)$ for every $i, j \in V$ and the respective architecture is an undirected graph.

There are various computational modes possible for recurrent networks depending on the choice of sets α_t of updated units in rule (2.2). In particular, a recurrent network is said to be in a *sequential* mode if for every time instant $t \geq 1$ at most one unit updates its state according to (2.2), that is $|\alpha_t| \leq 1$. In order to formally avoid long constant intermediate computations when only those units are updated that do actually not change their outputs, a so called *productive* computation of length $t^\star$ discrete updates is introduced so that for every $1 \leq t \leq t^\star$ there exists a unit $j \in \alpha_t$ updated at time t such that $y_j^{(t)} \neq y_j^{(t-1)}$. Moreover, it is said that a productive computation of a recurrent network *terminates*, *converges* or *reaches a stable state* $\mathbf{y}^{(t^\star)}$ at time $t^\star \geq 0$ if $\mathbf{y}^{(t^\star)} = \mathbf{y}^{(t^\star+k)}$ for all $k \geq 1$ (or for analog networks, at least $\|\mathbf{y}^{(t^\star)} - \mathbf{y}^{(t^\star+k)}\| \leq \varepsilon$ holds for some small constant $0 \leq \varepsilon < 1$). Usually, some *systematic* choice of α_t is employed, e.g. $\alpha_{\tau s+j} = \{j\}$ for $j = 1, \ldots, s$ where a discrete *macroscopic time* $\tau = 0, 1, 2, \ldots$ is introduced during which all the units in the network are updated (in general case some of them may update their outputs even several times). For a *parallel* mode, on the other hand, there exists a time instant $t \geq 1$ with $|\alpha_t| \geq 2$ when at least two neurons re-compute their new outputs simultaneously. Typically, a *fully parallel* mode is considered in which all the units are updated at each time instant, that is $\alpha_t = V$ for every $t \geq 1$.

Furthermore, in *asynchronous* computations the respective choice of α_t is random and each single unit in the network may in fact decide independently at which time instant its state is updated. To the contrary, in *synchronous* computations set α_t is determined deterministically at any time instant t and its choice is centrally driven. The seemingly less powerful asynchronous model have been shown for binary states to have the same computational power as its synchronous systematic counterpart at the cost of only a small overhead in the size and time of the (simulating) asynchronous network for both sequential and parallel modes and even for symmetric networks [90]. Hence, a synchronous model will mostly be assumed.

For the purpose of universal computations over inputs of arbitrary lengths, different input protocols have been introduced. Thus in the following both the recurrent networks with finite number of units and the infinite families of cyclic networks, each for one input length, are considered.

2.1.3.1 Finite Recurrent Networks The computational power of recurrent networks has been studied by the analogy with the traditional models of computations so that the networks are exploited as acceptors of languages $L \subseteq \{0,1\}^\star$ over the binary alphabet. In the finite, so called *neural acceptors* working under fully parallel update, an input string $\mathbf{x} = x_1 \dots x_n \in \{0,1\}^n$ of arbitrary length $n \geq 0$ is sequentially presented to the network bit after bit via an input neuron $inp \in V$ whose state is externally set to the respective input bits at prescribed time instants regardless of an influence from the remaining units in the network. The output neuron $out \in V$ subsequently signals whether the input string $\mathbf{x}$ belongs to the underlying language L. Especially in the *binary-state* neural acceptors these input bits x_i $(i = 1, \dots, n)$ are presented with a given *period* of $p \geq 1$ discrete steps, i.e. $y_{inp}^{(p(i-1))} = x_i$ and the output neuron recognizes each prefix of input string online possibly with some time delay $k \geq 1$, that is $y_{out}^{(p(i-1)+k+1)} = 1$ iff $x_1 \dots x_i \in L$. It has been shown for binary networks that the constant time delay k can always be reduced to 1 within only a linear increase in the network size [117]. Or in the *analog-state* networks usually $p = 1$ and an additional *validation* input unit $ival \in V$ is employed to indicate the end of an input string, i.e. $y_{ival}^{(t)} = 1$ for $t = 0, \dots, n-1$ and $y_{ival}^{(t)} = 0$ for $t \geq n$, while the output neuron after some time $T(n) \geq n$ that depends on the input length n provides the result of recognition which is again announced by a validation output unit $oval \in V$, that is $y_{oval}^{(T(n))} = 1$, and $y_{out}^{(T(n))} = 1$ iff $\mathbf{x} \in L$, whereas $y_{out}^{(t)} = y_{oval}^{(t)} = 0$ for $t \neq T(n)$. Moreover, in analog networks an alternative input protocol is also possible since the input string $\mathbf{x}$ of arbitrary length n can be encoded into a real initial state of input neuron, e.g. $y_{inp}^{(0)} = \sum_{i=1}^{n} (2x_i + 1)/4^i$ [112].

2.1.3.1.1 Asymmetric Recurrent Networks The computational power of finite recurrent neural networks with in general asymmetric weights employing the saturated-linear activation function (2.4) depends on the descriptive complexity of weights. For integer weights these models coincide with finite **binary recurrent networks** of the threshold gates with activation function (2.3). Obviously, the computational power of binary neural acceptors which have clearly only a finite number of network states corresponds to that of finite automata [57] and hence they can shortly

be called *neuromata* recognizing exactly the class of regular languages. Furthermore, finer descriptive measures have been studied [5] to find out how are the neural implementations of finite automata efficient. It has been shown that a neuromaton of size $O(\sqrt{q})$ can be constructed which, for the constant period $p = 4$ of presenting the input bits, simulates a given deterministic finite automaton with q states [53, 54] and in the worst case, this size $\Omega(\sqrt{q})$ cannot be reduced if either the period is at most $p = O(\log q)$ [53] or polynomial weights are assumed [54]. Moreover, the size of neuromaton can also be compared to the length of regular expressions whose descriptional efficiency may exceed that of deterministic finite automata. In particular, an optimal-size neuromaton with $\Theta(\ell)$ units has been constructed recognizing a regular language described by a given regular expression of length ℓ [117]. Neuromata, on the other hand, have been shown to describe some regular languages that require regular expressions of exponential length, with only a linear network size [117] which confirms their powerful descriptive capabilities.

The **finite analog recurrent networks with rational weights** employing the saturated-linear activation function (2.4) can simulate arbitrary Turing machines step per step [112]. Hence, by implementing a universal Turing machine, any function computable by a Turing machine in time $T(n)$ can be computed by a fixed analog recurrent network with only 886 units in time $O(T(n))$ [112]. The size of this universal network has further been reduced even to 25 neurons at the cost of simulating time $O(n^2T(n))$ [54]. It follows that polynomial-time computations of such networks correspond to the well-known complexity class P. In addition, the Turing universality has been shown for a more general class of activation functions [60] including the standard sigmoid (2.5) [56] although the respective simulations require exponential-time overhead per each computational step.

Furthermore, the **finite analog recurrent networks with arbitrary real weights** and the saturated-linear activation function (2.4) have even super-Turing computational capabilities. In particular, the computational power of such analog neural acceptors working within time $T(n)$ is exactly the same as that of infinite (nonuniform) families of threshold circuits whose size is polynomially related to $T(n)$ [111]. This means that polynomial-time computations of such networks correspond to the nonuniform complexity class P/poly (see [7] for definition) and within exponential time any input-output mapping can be computed. Moreover, a proper hierarchy of nonuniform complexity classes between P and P/poly has been proved for polynomial-time computations of analog recurrent networks with increasing Kolmogorov complexity (information contents) of real weights [8]. For example, setting a logarithmic bound on the resource-bounded Kolmogorov complexity of the real weights, the languages recognized correspond to the complexity class Full-P/log (see [9] for definition).

All the preceding results concerning analog computations of finite recurrent networks assume arbitrary-precision real number calculations. However, any amount of **analog noise** reduces the computational power of analog recurrent networks to that of neuromata [15, 77]. In this case, the above-mentioned upper bounds on the neuromaton size remain valid for a general class of activation functions [77, 107, 114]. Further, for common noise distributions that are nonzero on large sets such networks

are even unable to recognize all regular languages [80, 110] (e.g. they recognize definite languages [99]).

Besides their computational power the complexity of related problems have been studied for finite recurrent networks. For example, the issue of whether there exists a stable state in a given binary network was proved to be NP-complete [3, 30, 64, 97]. Or the **halting problem** which is the issue of whether a recurrent network terminates its computation over a given input was shown to be PSPACE-complete for binary networks [25, 63, 97], and algorithmically undecidable for analog networks with rational weights and only 25 units which follows from their computational universality [54]. Note also that the computations of recurrent networks of size s that terminate within time $t^\star$ can be unwound into circuits of size $st^\star$ and depth $t^\star$ by standard technique [105] which, for example, implies the computational equivalence of feedforward and *convergent* recurrent networks [34].

2.1.3.1.2 Symmetric Recurrent Networks The well-known fundamental property of symmetric (Hopfield) networks is that their dynamics are constrained by a *Liapunov*, or *energy* function E which is a bounded function defined on their state space decreasing along any productive computation. It follows from the existence of such a function that the network state converges towards some stable state corresponding to a local minimum of E. In the binary symmetric nets, sequential computations starting from any initial state terminate provided that $w(j,j) \geq 0$ for every $j \in V$ (so called *semisimple* networks) [49] which can be proved by using e.g. the following energy function

$$E(\mathbf{y}) = -\frac{1}{2} \sum_{j=1}^{s} \sum_{i=0}^{s} w_{ji} y_i y_j \,. \tag{2.7}$$

The parallel computations of binary Hopfield nets were shown either to reach a stable state (e.g. when quadratic form (2.7) is negative definite [36, 37]) or to eventually alternate between two different states [14, 36, 96, 127]. These **convergence** results were further generalized for analog symmetric networks which are proved under mild hypotheses to converge to a fixed point or to a limit cycle of length at most two for parallel updates [27, 59] by applying the Liapunov function of the form:

$$E(\mathbf{y}) = -\frac{1}{2} \sum_{j=1}^{s} \sum_{i=0}^{s} w_{ji} y_i y_j + \sum_{j=1}^{s} \int_0^{y_j} \sigma^{-1}(y) dy \,. \tag{2.8}$$

The **convergence time** in Hopfield nets, i.e. the number of discrete-time updates before the network converges (within a given precision for analog states), have also been studied. In symmetric networks of s binary neurons a trivial 2^s upper bound holds since there are only 2^s different network states. In the worst case, the convergence time of Hopfield nets may indeed be exponential for both sequential [43] and parallel [40] modes. This is witnessed e.g. by symmetric, sequential or parallel implementations of a *binary counter* that traverses most of the network space before it converges in time $\Omega(2^{s/8})$ asynchronous sequential updates [42] or $\Omega(2^{s/3})$ fully parallel steps [38, 39]. For the average case, on the other hand, a very fast convergence $O(\log\log s)$ of binary

Hopfield nets can be shown under reasonable assumptions [61]. However, the previous bounds do not take into account the size of the *weights*. An upper bound of $O(W)$ on the convergence time of binary sequential [26, 35, 37, 39] and parallel [36] Hopfield nets where

$$W = \sum_{j,i \in V} |w_{ji}| \tag{2.9}$$

is a so called *weight* of the network follows from the characteristics of the energy function (2.7). For integer weights (for real weights see [26, 37]), this upper bound can be expressed more precisely as $(\sum_{j=1}^{s} \sum_{i=1;i\neq j}^{s} |w_{ji}| + \sum_{j=1}^{s} |w_{j0}+e_j|)/(2+2\min_{j\in V} w_{jj})$ for sequential mode, or $(\sum_{j=1}^{s} \sum_{i=1}^{s} |w_{ji}| + 3\sum_{j=1}^{s} |w_{j0}+e_j| - s)/2$ for parallel updates where $e_j = 1$ if $\sum_{i=0}^{s} w_{ji}$ is even, and $e_j = 0$ otherwise [22]. This yields a polynomial-time convergence for binary symmetric networks with polynomial weights. Moreover, these results may further be translated into convergence time bounds with respect to the full descriptional complexity of Hopfield nets, i.e the number of bits in their representations. For binary symmetric networks which are described with M bits, convergence-time lower and upper bounds $2^{\Omega(M^{1/3})}$ and $2^{O(M^{1/2})}$, respectively, have been shown [116]. This can be compared to the convergence-time result for analog Hopfield nets in which the precision of real weight parameters plays an important role. In particular, the corresponding lower bound $2^{\Omega(g(M))}$ updates have been obtained, where $g(M)$ is an arbitrary continuous function such that $g(M) = \Omega(M^{2/3})$, $g(M) = o(M)$, and $M/g(M)$ is increasing, which provides an example of the analog Hopfield net whose computation terminates later than that of any other binary symmetric network of the same representation size [116].

As the Hopfield networks have originally been used as associative memories the issue of determining the number of **stable states** corresponding to patterns stored in the Hopfield memory, is important. It has been shown that in average there are asymptotically $1.05 \times 2^{0.2874s}$ many stable states in binary Hopfield nets of size s, with zero feedbacks and biases ($w_{jj} = w_{j0} = 0$ for $j \in V$) whose other weights are independent identically zero-mean Gaussian random variables [82, 126]. For a particular binary symmetric network, however, the issue of deciding whether there are e.g. at least one (when negative feedback weights are allowed) [23], two [64], or three [23] stable states, is NP-complete. Indeed, the problem of determining the exact number of stable states for a given binary Hopfield net is #P-complete [23, 64] (see [29] for definition of #P). Also the issue of finding a stable state in a binary symmetric network was shown to be complete (by LOGSPACE reductions) for the complexity class of polynomial-time local search problems [106], i.e. it is PLS-complete (see [55] for definition of PLS). Furthermore, the problem of *attraction radius* of a stable state, i.e. how many binary outputs may be flipped in a given stable network state so that the respective sequential or fully parallel Hopfield net still converges to it, is NP-hard [24]. It was even shown that there is no polynomial-time algorithm that approximates the attraction radii in a sequential or fully parallel binary Hopfield net within a factor $s^{1-\varepsilon}$ for any fixed $0 < \varepsilon \leq 1$ (unless $P = NP$) [24].

Hopfield networks have also been alternatively applied to the fast approximate solution of combinatorial optimization problems [51]. The cost function of an optimization problem is here encoded into the energy of Hopfield network which is minimized in

the course of computation. Hence, the issue of finding a network state with **minimal energy** or energy less than a prescribed value for a given Hopfield net is of special interest (so called *MIN ENERGY problem*). However, this problem is in general NP-complete for both binary [10, 11] and analog [116] Hopfield nets. On the other hand, for binary Hopfield nets whose architectures are planar lattices [13] or planar graphs [10] this issue is polynomial-time solvable. Further, a polynomial-time *approximate* algorithm that solves the MIN ENERGY problem within absolute error less than $0.243W$ in binary Hopfield nets of weight W have been introduced [116] which is based on high-performance approximate algorithm for MAX CUT problem [31, 81]. For $W = O(s^2)$, e.g. for symmetric networks with s binary neurons and constant weights, this results matches the lower bound $\Omega(s^{2-\varepsilon})$ which cannot be guaranteed by any approximate polynomial-time MIN ENERGY algorithm for every $\varepsilon > 0$ (unless $P = NP$) [11]. In addition, MIN ENERGY can be approximated within absolute error $O(s/\log s)$ in polynomial time for special binary Hopfield nets whose architectures are two-level grids [12].

Finally, results on the **computational power** of Hopfield nets will be surveyed similarly as for asymmetric recurrent networks in Paragraph 2.1.3.1.1. In fact, for binary networks the computational equivalence between *convergent* asymmetric networks and Hopfield nets was shown within only a linear overhead in time and size of the simulating symmetric network [116] which is an improvement of the original quadratic-size construction [89]. This tight converse to Hopfield's convergence theorem [49] thus shows that not only do all binary symmetric network converge, but all convergent computations can be implemented efficiently in Hopfield nets. This had previously been known for threshold circuits that can be implemented by using only symmetric interconnection within the same architecture [93]. However, the symmetric binary neural acceptors for arbitrary-length input sequences are properly weaker than finite automata [44] and recognize a strict subclass of the regular languages, so called *Hopfield languages* [117]. More precisely, it was proved in [113] that a regular language $L \subseteq \{0,1\}^\star$ is a Hopfield language iff for every prefix and suffix $\mathbf{v}_1, \mathbf{v}_2 \in \{0,1\}^\star$ and for any two-bit string $\mathbf{x} \in \{0,1\}^2$ there exists k_0 such that either $\mathbf{v}_1\mathbf{x}^k\mathbf{v}_2 \in L$ for every $k \geq k_0$ or $\mathbf{v}_1\mathbf{x}^k\mathbf{v}_2 \notin L$ for every $k \geq k_0$.

Also **analog symmetric neural acceptors** are known to faithfully recognize Hopfield languages [77, 114] which provides a lower bound on the computational power of finite analog Hopfield nets. Because of the above-mentioned convergence property [59] the finite analog Hopfield nets are unlikely to simulate arbitrary Turing machines. On the other hand, if the fully parallel analog symmetric networks are augmented with an external oscillator producing any infinite binary sequence that contains infinitely many three-bit substrings of the form $bx\bar{b} \in \{0,1\}^3$ where $b \neq \bar{b}$ then such devices can be exploited for simulating any given analog asymmetric recurrent network within linear-size and the same Kolmogorov complexity of real weights [116]. This implies that the results on the computational power of analog asymmetric networks are still valid for analog Hopfield nets with an external oscillator of certain type, especially for rational weights these devices are Turing universal.

2.1.3.2 Infinite Families of Binary Recurrent Networks In an alternative input protocol, infinite families $\{N_n\}$ of binary recurrent networks, each N_n for one input length $n \geq 0$, are exploited for universal computations by the analogy with circuit complexity. Thus, for n-bit binary inputs $\mathbf{x} \in \{0,1\}^n$ the network N_n is used whose n input neurons are initialized accordingly within the initial network state. For recognizing a language $L \subseteq \{0,1\}^\star$, the respective network N_n is employed for an input string $\mathbf{x} \in \{0,1\}^n$ and, after it converges in time $t^\star$, its single output neuron *out* is read which indicates whether $\mathbf{x}$ belongs to L, that is $y_{out}^{(t^\star)} = 1$ iff $\mathbf{x} \in L$. Within this context the *size* $S(n)$ can be defined as the number of units in N_n. It has been known that the polynomial-size families of binary recurrent networks recognize exactly the languages in the complexity class PSPACE/poly [63] (see [7] for definition). The above-mentioned equivalence between convergent asymmetric and symmetric networks gives the same result for the families of Hopfield nets, i.e. they also recognize exactly PSPACE/poly [89]. In addition, if the Hopfield nets in these families are restricted to have polynomial weights (with respect to the input length) then their computational power reduces to P/poly [89].

2.1.4 Probabilistic Networks

The computational properties of various stochastic versions of discrete-time perceptron networks have also been studied. The reference model of *probabilistic (stochastic) networks* can be defined so that the respective deterministic model is augmented with additional random *binary* input units $i \in \Pi$ whose states in time establish independent identically distributed binary sequences, that is for all discrete time instants $t \geq 0$ the probability of $y_i^{(t)} = 1$ is given by a real value p_i $(0 \leq p_i \leq 1)$, and consequently $y_i^{(t)} = 0$ with probability $1 - p_i$ $(i \in \Pi)$. This model of probabilistic networks can usually be shown polynomially (in parameters) related to neural networks with other stochastic behavior including those which are unreliable in computing states and connecting units [109, 128]; Boltzmann machines [1, 94, 95], etc. The probabilistic networks are also exploited for language recognition in a similar way as their deterministic counterparts in order to analyze their computational power. Thus, a language $L \subseteq \{0,1\}^\star$ is *ε-recognized* with probability $0 < \varepsilon < 1/2$ of an error, that is the probability of a network outputs 1 for input $\mathbf{x} \in \{0,1\}^\star$ is at least $1 - \varepsilon$ if $\mathbf{x} \in L$ and at most ε for $\mathbf{x} \notin L$. Note, that the previous symmetry in the probability of errors in accepting and rejecting an input can be achieved in general case, e.g. by adding few random input units [41].

2.1.4.1 Probabilistic Feedforward Networks For the simplicity consider first the probabilistic binary-state feedforward networks (probabilistic threshold circuits). The language recognition with families of probabilistic single-output threshold circuits having a high probability of error (e.g. $\varepsilon = 0.4$) is not very reliable, however, the error can be reduced arbitrarily by repeating the computation in parallel. In particular, any language that is ε-recognized $(0 < \varepsilon < 1/2)$ by probabilistic feedforward networks of size s units and d layers can be λ-recognized by a depth-(d+1) stochastic threshold circuits with $\lceil 2 \log_{4\varepsilon(1-\varepsilon)} \lambda \rceil s + 1$ gates for any $0 < \lambda < \varepsilon$. Therefore, any probabilistic

binary feedforward network can even be replaced with a reasonable large equivalent *deterministic* threshold circuit. Indeed, it was proved in [95] that for any language $L \subseteq \{0,1\}^n$ that is ε-recognized ($1/4 < \varepsilon < 1/2$) by a probabilistic feedforward network of size s units, n inputs and d layers there exists a depth-(d+1) (deterministic) threshold circuit of size $\lceil 8\varepsilon \ln 2/(1-2\varepsilon)^2 + 1 \rceil ns + 1$ recognizing L. Also, the complexity classes TC_d^0 ($d \geq 1$) associated with the families of threshold circuits have been generalized to their probabilistic versions RTC_d^0 which are the classes of all the languages $\varepsilon(n)$-recognized by the families of polynomial-size and polynomial-weight probabilistic threshold circuits of depth d with an error given by a real sequence of probabilities $\varepsilon(n) = 1/2 - 1/n^{O(1)}$, each for one input length $n \geq 0$ [41]. For example, language IP that consists of all the input instances for which the Boolean inner product gives 1, has been shown to belong to RTC_2^0 [41] confirming that probabilistic feedforward networks may be more efficient than their deterministic counterparts since as we already know $IP \notin TC_2^0$. On the other hand, it follows from the preceding that only one layer can be saved by introducing the stochasticity in the threshold circuits since it holds $RTC_d^0 \subseteq TC_{d+1}^0$ for every $d \geq 1$ (for nonuniform circuit families) [41].

2.1.4.2 Probabilistic Recurrent Networks The computational analysis have also been generalized for finite probabilistic recurrent networks with linear-saturated activation function (2.4) [109]. For integer weights, the results coincides with that for deterministic networks (see Paragraph 2.1.3.1.1), that is the binary-state probabilistic networks ε-recognize the regular languages. Further, the analog-state probabilistic networks with rational weights can ε-recognize in polynomial time exactly the languages from the nonuniform complexity class Pref-BPP/log which corresponds to polynomial-time bounded-error probabilistic Turing machines with prefix-closed logarithmic advice functions (see [7, 9] for precise definitions). The weak super-Turing computational capability comes here from the probabilities p_i associated with random input units $i \in \Pi$ which are allowed to be *arbitrary* real numbers. On the other hand, if one restricts these probabilities only to rational numbers, the computational power of polynomial-time analog-state probabilistic networks with rational weights reduces to the recursive complexity class BPP. Finally, for arbitrary real weights the computational power of probabilistic recurrent networks working in time $T(n)$ is polynomially related to that of the corresponding deterministic networks within simulating time $nT(n) + n^2$ which implies that e.g. polynomial-time computations correspond to the complexity class P/poly.

2.2 Continuous-Time Dynamics

In the continuous-time neural networks the dynamics of analog network state $\mathbf{y}(t) \in \Re^s$ is defined for every real $t > 0$ usually as the solution of a system of s differential equations, each equation for one continuous-time unit. The corresponding boundary conditions are given by an initial network state $\mathbf{y}(0)$. For example, consider the following system:

$$\frac{dy_j}{dt}(t) = -y_j(t) + \sigma(\xi_j(t)) = -y_j(t) + \sigma\left(\sum_{q=0}^{s} w_{ji} y_j(t)\right) \qquad j = 1, \ldots, s \qquad (2.10)$$

where the excitation $\xi_j(t)$ of unit j is defined as in (2.1) and the saturated-linear activation function (2.4) is employed implying $\mathbf{y}(t) \in [0,1]^s$. By Liapunov function (2.8) it can be shown that a continuous-time symmetric network (with $w_{ji} = w_{ij}$) converges from any initial state $\mathbf{y}(0)$ to some stable state satisfying $dy_j/dt = 0$ for all $j = 1, \ldots, s$ [17, 50]. Also an exponential lower bound on the convergence time of continuous-time Hopfield nets has been obtained in terms of the system dimension [115]. The set of stable states of the discrete system (2.2) coincides with that of the continuous-time system (2.10). Moreover, a continuous-time asymmetric network controlled by dynamics (2.10) can be constructed that simulates a given finite binary-state discrete-time recurrent network faithfully within only a linear-size overhead [91]. Such a simulation has even be achieved with the continuous-time Hopfield nets [115]. This implies that the polynomial-size infinite families of continuous-time (symmetric) networks are able to recognize at least the complexity class PSPACE/poly (see Paragraph 2.1.3.2).

3 Other Neural Network Models

Also the computational capabilities of neural network models that are alternative to the perceptron networks have recently been analyzed. In the following sections, the RBF, winner-take-all, and spiking networks will be reviewed.

3.1 RBF Networks

Units in the *RBF networks* that represent an important alternative to the classical discrete-time perceptron networks compute the so-called *radial basis functions* which were first introduced in the context of interpolation problems [98]. Thus, the perceptron update rule (2.1), (2.2) for a unit $j \in V$ is now modified so that the respective "excitation"

$$\xi_j^{(t)} = \frac{\left\| \mathbf{x}_j^{(t)} - \mathbf{w}_j \right\|}{w_{j0}} \qquad (3.1)$$

is proportional to the distance between the corresponding input vector $\mathbf{x}_j^{(t)} = (y_{ji_1}^{(t)}, \ldots, y_{ji_{n_j}}^{(t)}) \in \Re^{n_j}$ that consists of states of incoming units $i_1, \ldots, i_{n_j} \in V$ incident to j, and its so called *center* $\mathbf{w}_j = (w_{ji_1}, \ldots, w_{ji_{n_j}}) \in \Re^{n_j}$ which is composed of the corresponding "weights", where the real positive "bias" parameter $w_{j0} > 0$ determines its so called *width*. The new state of RBF unit j at the next time instant $t+1$ is determined similarly as in (2.2) but with an activation function $\sigma : \Re \longrightarrow \Re$, e.g. the *Gaussian function* $\sigma(\xi) = e^{-\xi^2}$, whose shape usually differs from that of sigmoid functions (2.3), (2.4), and (2.5). For representing the binary values 0 and 1, two different analog states of RBF units are reserved. Within this protocol the universal Boolean

NAND gate over multiple literals (input variables or their negations) can be implemented by the RBF unit employing the maximum norm ($\|\mathbf{x}\| = \|\mathbf{x}\|_\infty = \max_{i=1,\ldots,n} |x_i|$ for $\mathbf{x} = (x_1, \ldots, x_n) \in \Re^n$) in (3.1). For a large class of activation functions with a special type of inflexion including the Gaussian function, this NAND implementation is robust even when the analog representation of binary values is released to be within specific disjoint small intervals around the respective real states [125]. It follows that many results described in Section 2 concerning the computational power of perceptron networks can be reformulated also for the RBF networks. For example, the deterministic finite automata with q states can be implemented by recurrent networks with $O(\sqrt{q \log q})$ RBF units in a robust way [125]. On the other hand, the Turing universality of the finite RBF networks represents still an open problem.

3.2 Winner-Take-All Networks

Another neural network model whose computational power has recently been studied consists of so called *winner-take-all* (WTA) gates that employ, in neurocomputing quite common, competitive strategy used e.g. in Kohonen networks [58]. The competition principle appears to be neurophysiological plausible and has also efficient analog VLSI implementations [6]. Thus, a k-WTA_n gate with n real inputs and n binary outputs computes a mapping k-$WTA_n : \Re^n \longrightarrow \{0,1\}^n$ defined as k-$WTA_n(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$ where the ith output gives $y_i = 1$ $(1 \leq i \leq n)$ iff the number of input values greater than x_i is at most $k-1$, i.e. $|\{j;\, x_j > x_i,\, 1 \leq j \leq n\}| \leq k-1$. In particular, a WTA_n gate for $k = 1$ (which is omitted in notation) indicates which of the n inputs has maximal value. However, even this simple WTA_n device $(n \geq 3)$ cannot be implemented by any perceptron network having less than $\binom{n}{2} + n$ threshold gates [73] which clearly suffice for WTA_n computation. This implies that any *recurrent* perceptron network with $O(n)$ threshold gates cannot compute WTA_n in sublinear time $o(n)$. Furthermore, any Boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$ in TC_0^2 (i.e. computed by two-layered networks of polynomial size and weights) can be computed by a single k-WTA_r gate applied to $r = O(n^c)$ (for some constant c) weighted sums (2.1) of n input variables with natural (positive) weights of polynomial size [73]. Or the winner-take-all networks may gain efficiency in the total wire length as the function P_{LR}^k introduced in Paragraph 2.1.2.1 can be computed by a two-layered network consisting of only two winner-take-all units (with weighted inputs) and one threshold gate, whose total wire length reduces to $O(k)$ [62]. Hence, it appears that the winner-take-all gates are more efficient than the perceptrons.

3.3 Networks of Spiking Neurons

The most prominent position among neural network models that are alternative to the classical perceptron paradigm is occupied by *networks of spiking neurons* (artificial *pulsed neural networks*) or, shortly, *spiking networks*. The spiking neurons are supposed to be more biologically plausible units than previous neuron models [74]. Their computational properties will first be surveyed for a deterministic model, and subsequently, also a noisy version will be inspected.

3.3.1 Deterministic Spiking Networks

The main difference of spiking networks from traditional neural network models consists in encoding the computational states by temporal differences between so-called *spikes*, or *firing times* of neurons. Thus, suppose a sequence $0 \leq y_j^{(1)} < y_j^{(2)} < \cdots < y_j^{(\tau)} < \ldots$ of firing times is associated with each spiking neuron $j \in V$. Denote by $Y_j(t) = \{y_j^{(\tau)} < t;\, \tau \geq 1\}$ the set of spikes of neuron j before a continuous time instant $t \geq 0$, while $y_j(t) = \max Y_j(t)$ denotes its last firing time for $Y_j(t) \neq \emptyset$. Formally choose $y_j^{(0)} < 0$ and define $y_j(t) = t$ for all $0 \leq t < y_j^{(1)}$ and every $j \in V$. The spikes for the input neurons $V_{inp} \subseteq V$ are given externally and encode the input. For example, a binary input string $\mathbf{x} = (x_1, \ldots, x_n) \in \{0,1\}^n$ can be presented either by n separate input neurons, or *online* (bit after bit) by a single input unit $inp \in V_{inp}$, so that each bit is represented by firing or nonfiring within a given time interval (e.g. after the firing of a designated neuron). Or this input is encoded by the temporal difference $y_{inp}^{(2)} - y_{inp}^{(1)} = \sum_{i=1}^{n} 2^{-i-c} x_i$ (for sufficiently large integer constant $c > 0$) between the only two spikes $y_{inp}^{(1)} < y_{inp}^{(2)}$ of input neuron inp.

In addition, for each connection from i to j a *response function* $\varepsilon_{ji} : \Re_0^+ \longrightarrow \Re$ ($\Re_0^+$ denotes the set of nonnegative real numbers) is introduced which models the generic response of neuron j to spikes from presynaptic unit i in continuous time $t \geq 0$. Due to notation simplicity, the response functions are formally defined to be constant zero for missing connections, that is $\varepsilon_{ji}(t) = 0$ for $t \geq 0$ when $w_{ji} = 0$. The weights w_{ji} associated with connections are assumed to be positive for all $j, i \in V$. Furthermore, for each non-input unit $j \in V \setminus V_{inp}$ a nonpositive *bias function* $w_{j0} : \Re_0^+ \longrightarrow \Re_0^- \cup \{-\infty\}$ determines its nonnegative threshold potential value $\Theta_j(t) = -w_{j0}(t) \geq 0$ at time $t \geq 0$ which, when being reached, causes neuron j to generate a spike as follows. Note that the function $\Theta_j = -w_{j0}$ opposite to the bias is usually called the *threshold function* which should not be confused with the hard limiter (2.3). The spikes $y_j^{(\tau)}$, $\tau \geq 1$ for each non-input neuron $j \in V \setminus V_{inp}$ are determined recursively:

$$y_j^{(\tau)} = \inf \left\{ t \geq 0;\, t > y_j^{(\tau-1)} \,\&\, \xi_j(t) \geq 0 \right\} \tag{3.2}$$

according to its excitation

$$\xi_j(t) = w_{j0}(t - y_j(t)) + \sum_{i=1}^{s} \sum_{y \in Y_i(t)} w_{ji} \cdot \varepsilon_{ji}(t - y) \tag{3.3}$$

defined for continuous time $t \geq 0$ which, excluding the bias term $w_{j0}(t - y_j(t))$, is usually called its *potential*. Finally, the spikes of output neurons encode the corresponding result of the computation, e.g. the same output protocol as for the input can be used.

The subsequent results concerning the computational power of spiking networks are valid for a very general class of response and bias functions satisfying the following relatively weak conditions [69]. Each **response function** ε_{ji} associated with connection from i to j is everywhere either positive $\varepsilon_{ji} \geq 0$ or negative $\varepsilon_{ji} \leq 0$ modeling a biological *excitatory* (EPSP) or *inhibitory* (IPSP) *postsynaptic potential*, respectively. Further, $\varepsilon_{ji}(t) = 0$ for $t \in [0, \Delta_{ji}]$ where $0 < \Delta_{\min} \leq \Delta_{ji} \leq \Delta_{\max}$ is the individual synaptic *delay*

associated with connection from i to j. The response functions ε_{ji} are *stereotyped* so that there exist some general functions $\varepsilon^E, \varepsilon^I : \Re_0^+ \longrightarrow \Re$ with $\varepsilon^E(t) = 0$ for $t \geq t_{end}^E$ and $\varepsilon^I(t) = 0$ for $t \geq t_{end}^I$ such that for every $j, i \in V$ and for all $t \geq 0$ the EPSP $\varepsilon_{ji}(\Delta_{ji} + t) = \varepsilon^E(t) \geq 0$ and the IPSP $\varepsilon_{ji}(\Delta_{ji} + t) = \varepsilon^I(t) \leq 0$. In addition, ε^E has a global maximum and there exist at least two short time-segments where ε^E is linearly increasing and decreasing, respectively. On the other hand, ε^I is negative in $(0, t_{end}^I)$, nonincreasing in $[0, t_1]$, and nondecreasing in $[t_2, t_{end}^I]$ for some $0 < t_1 < t_2 < t_{end}^I$. The examples of the EPSP functions include mathematically simple piecewise linear functions such as

$$\varepsilon_{ji}(t) = \begin{cases} 0 & \text{for } 0 \leq t \leq \Delta_{ji} \\ 1 - |t - \Delta_{ji} - 1| & \text{for } \Delta_{ji} < t < \Delta_{ji} + 2 \\ 0 & \text{for } t \geq \Delta_{ji} + 2\,. \end{cases} \tag{3.4}$$

Sometimes the strict monotony for the EPSP functions is even not required [78] and simple piecewise constant functions are employed, e.g.

$$\varepsilon_{ji}(t) = \begin{cases} 0 & \text{for } 0 \leq t \leq \Delta_{ji} \\ 1 & \text{for } \Delta_{ji} < t < t_{end}^E \\ 0 & \text{for } t \geq t_{end}^E\,. \end{cases} \tag{3.5}$$

The examples of the IPSP functions include e.g. $-\varepsilon_{ji}$ defined by the EPSP counterpart ε_{ji} in (3.4) or (3.5). It appears that the preceding general assumptions are also satisfied by more biologically plausible response functions.

A general **bias function** $w_0 : \Re_0^+ \longrightarrow \Re_0^- \cup \{-\infty\}$ is often employed so that $w_{j0}(t) = w_0(t)$ for every $j \in V \setminus V_{inp}$ and all $t \geq 0$. It is assumed $w_0(t) = -\infty$ for $t \in (0, t_{ref})$ in order to prevent neurons from firing in the *refractory period* t_{ref} following the previous spike, and it is negative constant $w_0(t) = -\vartheta < 0$ equal to an initial value $w_0(0) = -\vartheta$ after some time $t \geq t_{end}$. The examples of the bias functions include mathematically simple piecewise constant functions such as

$$w_{j0}(t) = \begin{cases} -\vartheta & \text{for } t = 0 \\ -\infty & \text{for } 0 < t < t_{ref} \\ -\vartheta & \text{for } t \geq t_{ref}\,. \end{cases} \tag{3.6}$$

The **lower bounds** on the computational power of spiking networks from [69] will first be reviewed. Thus, for a given feedforward network with s threshold gates of unbounded weights and depth d, a neural network of $O(s)$ spiking neurons (or a spiking network of polynomial size with bounded weights from $[0, 1]$) can be constructed that simulates any of its computations within a time interval of length $O(d)$. It follows that any deterministic finite automaton (and Mealy or Moore machines) with q states can be simulated by a spiking network with the constant period of presenting the input bits that has $O(\sqrt{q})$ neurons or $O(\sqrt{q} \cdot \log q)$ units with bounded weights from $[0, 1]$. Finally, one can build a neural network of finite number of spiking neurons with *rational weights* from $[0, 1]$ (when all the other parameters introduced in the assumptions on response and threshold functions are rational numbers) that simulates any (multiple-tape) Turing machine in linear time so that each computational step requires only

a fixed constant number of spikes. Furthermore, any input-output mapping can be computed by a finite spiking network with *arbitrary real weights*.

On the other hand, the computational power of spiking networks can completely be characterized by introducing the **upper bounds**. It has been proved in [67] that the computational power of finite spiking networks with any piecewise linear response- and threshold-functions (not necessarily satisfying the above assumptions) equals that of finite discrete-time analog recurrent perceptron networks with any piecewise linear activations functions (e.g. the activation functions (2.3), (2.4) together are universal in this context). These networks are further computationally equivalent to so called N-RAM's, which are random access machines with $O(1)$ registers working with arbitrary real numbers of bounded absolute values (see [67] for exact definition). In addition, the previous equivalence is achieved with linear-time pairwise simulations and is also valid if one restricts all the numerical parameters in the models to be rational numbers. It follows that spiking networks with any piecewise linear response functions can easily be programmed to perform efficiently basic operations on analog variables in temporal coding such as addition and multiplication with a constant.

However, if one restricts to the **piecewise constant response functions** [78], e.g. (3.5), which are easier to implement in hardware then the spiking networks with piecewise monotone and continuous threshold functions employing arbitrary real-valued parameters cannot carry out the addition and multiplication with a constant on arbitrary small differences in firing times between neurons with a bounded number of spikes. Indeed, spiking networks with piecewise constant response functions and piecewise linear threshold functions with *rational* parameters are for online binary-string inputs computationally equivalent to deterministic finite automata within a fixed number of spikes per one simulated step. Furthermore, for *arbitrary real* parameters these restricted spiking networks with online inputs can, in principle, simulate any Turing machine but, in general, *not* within polynomial number of spikes. It follows that the computational power of spiking networks depends on the shape of the postsynaptic potentials and reduces essentially for piecewise constant response functions.

The preceding results concerning digital computations on spiking networks are based on the forced synchronization mechanism for the spikes. However, the small temporal differences in the firing times may be exploited to encode additional analog information. Indeed, the spiking neurons with **different synaptic delays** Δ_{ji} employing piecewise constant response- and threshold-functions are more powerful than the perceptrons. In particular, the Boolean *coincidence-detection* function $CD_k : \{0,1\}^{2k} \longrightarrow \{0,1\}$ $(k \geq 1)$ which formalizes a simple pattern-matching task as $CD_k(x_1, \ldots, x_k, x'_1, \ldots, x'_k) = 1$ iff there is $1 \leq i \leq k$ such that $x_i = x'_i = 1$ can easily be implemented by a single spiking neuron whose inputs are suitably delayed. On the other hand, any feedforward perceptron network that computes CD_k requires at least $k/log(k+1)$ threshold gates (2.3), or either $\Omega(\sqrt{k})$ or $\Omega(k^{1/4})$ units with piecewise polynomial (e.g. (2.4)) or exponential (e.g. (2.5)) activation functions, respectively [72]. This result has further been generalized for analog inputs and arbitrary reasonable response functions. For example, the witnessing *element distinctness function* $ED_n : (\Re_0^+)^n \longrightarrow \{0,1\}$ $(n \geq 2)$ which is defined as $ED_n(x_1, \ldots, x_n) = 1$ if $x_i = x_j$ for some $1 \leq i < j \leq n$, $ED_n(x_1, \ldots, x_n) = 0$ if $|x_i - x_j| \geq 1$ for all

$1 \leq i < j \leq n$, and arbitrarily otherwise, can be computed by a single spiking neuron while any feedforward network computing ED_n requires $\Omega(n \log n)$ first-layer threshold gates or $(n-4)/2-1$ hidden units with sigmoid activation function (2.5) [72].

3.3.2 Noisy Spiking Networks

As the spiking networks were supposed to model the information processing in biological neurons it is quite natural to introduce the notion of noise since the deterministic model in Paragraph 3.3.2 that assumes the precise firing times is not very realistic from that point of view. Thus, in the *noisy spiking networks* the excitation (3.3) of spiking neuron j just governs the probability that unit j fires. In particular, there are two functions $L, U : \Re \times \Re_0^+ \longrightarrow [0,1]$ given so that $L(\Xi, \Delta)$ provides a *lower* bound and $U(\Xi, \Delta)$ determines an *upper* bound on the *probability* that neuron j fires during a time interval T of length $\Delta \geq 0$ in which $\xi_j(t) \geq \Xi$ respectively $\xi_j(t) \leq \Xi$ for all $t \in T$ up to the next spike of j. It is assumed that L, U are nondecreasing in each of their two arguments (for any fixed value of the other argument) and $\lim_{\Xi \to -\infty} U(\Xi, \Delta) = 0$ for any fixed $\Delta > 0$ and $\lim_{\Xi \to \infty} L(\Xi, \Delta) > 0$ for any fixed $\Delta \geq t_1/6$ where $t_1 > 0$ determines the interval $[0, t_1]$ in which the function ε^E is now assumed to be initially nondecreasing. In addition, there exists a constant $\delta > 0$ such that $\varepsilon^E(t_1/6 + t) \geq \varepsilon^E(t) + \delta$ for all $t \in [0, 2t_1/3]$. In this case, the function $-\varepsilon^I$ satisfies the same conditions as ε^E. Finally, there exists a source of negative *background noise* that contributes to the excitation (3.3) of spiking neuron j an additive term that deviates for an arbitrarily long time interval by an arbitrarily small percentage from its average optional value $w_j^- \leq 0$.

For digital computations, it has been shown in [68] that under the previous weak conditions any Boolean function can be implemented by a sufficiently large network of noisy spiking neurons with arbitrarily high probability of correctness. Furthermore, for any deterministic finite automaton one can construct a network of noisy spiking neurons that simulates its computations of any given length with arbitrarily high probability of correctness [68]. Similar results have been achieved for analog computations [71, 75, 76], e.g. the noisy spiking networks with temporal encoding of analog values can reliably simulate the feedforward perceptron networks with real inputs and outputs [71].

Bibliography

[1] D.H. Ackley, G.E. Hinton, and T.J. Sejnowski, A learning algorithm for Boltzmann machines, *Cognitive Science* **9** (1) (1985) 147–169.

[2] E. Allender, A note on the power of threshold circuits, in: *Proceedings of the 30th Annual Symposium on Foundations of Computer Science FOCS'89, Research Triangle Park, North Carolina* (IEEE Computer Society Press, New York, 1989) 580–584.

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

[4] N. Alon and J. Bruck, Explicit construction of depth-2 majority circuits for comparison and addition, *SIAM Journal on Discrete Mathematics* **7** (1) (1994) 1–8.

[5] N. Alon, A.K. Dewdney, and T.J. Ott, Efficient simulation of finite automata by neural nets, *Journal of the ACM* **38** (2) (1991) 495–514.

[6] M.A. Arbib, *The Handbook of Brain Theory and Neural Networks* (The MIT Press, Cambridge, MA, 1995).

[7] J.L. Balcázar, J. Díaz, and J. Gabarró, *Structural Complexity I* (2nd edition, Springer–Verlag, Berlin, 1995).

[8] J.L. Balcázar, R. Gavaldà, and H.T. Siegelmann, Computational power of neural networks: A characterization in terms of Kolmogorov complexity, *IEEE Transactions of Information Theory* **43** (4) (1997) 1175–1183.

[9] J.L. Balcázar and M. Hermo, and E. Mayordomo, The structure of logarithmic advice complexity classes, *Theoretical Computer Science* **207** (1) (1998) 217–244.

[10] F. Barahona, On the computational complexity of Ising spin glass models, *Journal of Physics A: Mathematical and General* **15** (10) (1982) 3241–3253.

[11] A. Bertoni and P. Campadelli, On the approximability of the energy function, in: *Proceedings of the 4th International Conference on Artificial Neural Networks ICANN'94, Sorrento, Italy* (Springer–Verlag, Berlin, 1994) 1157–1160.

[12] A. Bertoni, P. Campadelli, C. Gangai, and R. Posenato, Approximability of the ground state problem for certain Ising spin glasses, *Journal of Complexity* **13** (3) (1997) 323–339.

[13] I. Bieche, J.P. Uhry, R. Maynard, and R. Rammal, On the ground states of the frustration model of a spin glass by a matching method of graph theory, *Journal of Physics A: Mathematical and General* **13** (8) (1980) 2553–2576.

[14] J. Bruck and J.W. Goodman, A generalized convergence theorem for neural networks, *IEEE Transactions of Information Theory* **34** (5) (1988) 1089–1092.

[15] M. Casey, The dynamics of discrete-time computation, with application to recurrent neural networks and finite state machine extraction, *Neural Computation* **8** (6) (1996) 1135–1178.

[16] A.K. Chandra, L.J. Stockmeyer, and U. Vishkin, Constant depth reducibility, *SIAM Journal on Computing* **13** (2) (1984) 423–439.

[17] M.A. Cohen and S. Grossberg, Absolute stability of global pattern formation and parallel memory storage by competitive neural networks, *IEEE Transactions on Systems, Man, and Cybernetics* **13** (5) (1983) 815–826.

[18] 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.

[19] T.M. Cover, Capacity problems for linear machines, in: L. Kanal, ed., *Pattern Recognition* (Thompson Book Co., 1968), 283–289.

[20] B. DasGupta and G. Schnitger, The power of approximating: A comparison of activation functions, in: S.J. Hanson, J.D. Cowan, and C.L. Giles, eds., *Advances in Neural Information Processing Systems* **5**, *NIPS'92* (Morgan Kaufmann, San Mateo, 1993) 615–622.

[21] B. DasGupta and G. Schnitger, Analog versus discrete neural networks, *Neural Computation* **8** (4) (1996) 805–818.

[22] P. Floréen, Worst-case convergence times for Hopfield memories, *IEEE Transactions of Neural Networks* **2** (5) (1991) 533–535.

[23] P. Floréen and P. Orponen, On the computational complexity of analyzing Hopfield nets, *Complex Systems* **3** (6) (1989) 577–587.

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

[25] P. Floréen and P. Orponen, Complexity issues in discrete Hopfield networks, Research Report A–1994–4, Department of Computer Science, University of Helsinki, 1994.

[26] F. Fogelman-Soulié, E. Goles-Chacc, and G. Weisbuch, Transient length in sequential iterations of threshold functions, *Discrete Applied Mathematics* **6** (1) (1983) 95–98.

[27] F. Fogelman-Soulié, C. Mejia, E. Goles-Chacc, S. Martínez, Energy function in neural networks with continuous local functions, *Complex Systems* **3** (3) (1989) 269–293.

[28] M. Furst, J.B. Saxe, and M. Sipser, Parity, circuits and the polynomial-time hierarchy, *Mathematical Systems Theory* **17** (1) (1984) 13–27.

[29] 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).

[30] G.H. Godbeer, J. Lipscomb, M. Luby, On the computational complexity of finding stable state vectors in connectionist models (Hopfield nets), Technical Report 208/88, Department of Computer Science, University of Toronto, 1988.

[31] M.X. Goemans and D.P. Williamson, Improved approximate algorithms for maximum cut and satisfiability problems using semidefinite programming, *Journal of the ACM* **42** (6) (1995) 1115–1145.

[32] M. Goldmann, J. Håstad, and A. Razborov, Majority gates vs. general weighted threshold gates, *Computational Complexity* **2** (4) (1992) 277–300.

[33] M. Goldmann and M. Karpinski, Simulating threshold circuits by majority circuits, *SIAM Journal on Computing* **27** (1) (1998) 230–246.

[34] L.M. Goldschlager and I. Parberry, On the construction of parallel computers from various bases of Boolean functions, *Theoretical Computer Science* **43** (1986) 43–48.

[35] E. Goles-Chacc, Dynamics of positive automata networks, *Theoretical Computer Science* **41** (1985) 19–32.

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

[37] E. Goles-Chacc, F. Fogelman-Soulié, and D. Pellegrin, Decreasing energy functions as a tool for studying threshold networks, *Discrete Applied Mathematics* **12** (3) (1985) 261–277.

[38] E. Goles-Chacc and S. Martínez, Exponential transient classes of symmetric neural networks for synchronous and sequential updating, *Complex Systems* **3** (6) (1989) 589–597.

[39] E. Goles-Chacc and S. Martínez, *Neural and Automata Networks: Dynamical Behavior and Applications* (Kluwer Academic Publishers, Dordrecht, 1990).

[40] E. Goles-Chacc and J. Olivos A., The convergence of symmetric threshold automata, *Information and Control* **51** (2) (1981) 98–104.

[41] A. Hajnal, W. Maass, P. Pudlák, M. Szegedy, and G. Turán, Threshold circuits of bounded depth, *Journal of Computer and System Sciences* **46** (2) (1993) 129–154.

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

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

[44] R. Hartley and H. Szu, A comparison of the computational power of neural network models, in: *Proceedings of the IEEE First International Conference on Neural Networks, San Diego* (IEEE Press, New York, 1987) 15–22.

[45] J. Håstad, On the size of weights for threshold gates, *SIAM Journal on Discrete Mathematics* **7** (3) (1994) 484–492.

[46] T. Hegedüs and N. Megiddo. On the geometric separability of Boolean functions, *Discrete Applied Mathematics* **66** (3) (1996) 205–218.

[47] T. Hofmeister, W. Hohberg, and S. Köhling, Some notes on threshold circuits, and multiplication in depth 4, *Information Processing Letters* **39** (4) (1991) 219–225.

[48] T. Hofmeister and P. Pudlák, A proof that division is not in TC^2_0, Research Report No. 447, Dortmund University, 1992.

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

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

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

[52] B.G. Horne and D.R. Hush, On the node complexity of neural networks, *Neural Networks* **7** (9) (1994) 1413–1426.

[53] B.G. Horne and D.R. Hush, Bounds on the complexity of recurrent neural network implementations of finite state machines, *Neural Networks* **9** (2) (1996) 243–252.

[54] P. Indyk, Optimal simulation of automata by neural nets, in: *Proceedings of the 12th Annual Symposium on Theoretical Aspects of Computer Science STACS'95*, (LNCS **900**, Springer–Verlag, Berlin, 1995) 337–348.

[55] D.S. Johnson C.H. Papadimitriou, and M. Yannakakis, How easy is local search?, *Journal of Computer and System Sciences* **37** (1) (1988) 79–100.

[56] J. Kilian and H.T. Siegelmann, The dynamic universality of sigmoidal neural networks, *Information and Computation* **128** (1) (1996) 48–56.

[57] S.C. Kleene, Representation of events in nerve nets and finite automata, in: C.E. Shannon and J. McCarthy, eds., *Automata Studies* (Vol. **34** of Annals of Mathematics Studies, Princeton University Press, NJ, 1956) 3–41.

[58] T. Kohonen, *Self-Organizing Maps* (3rd extended edition, Vol. **30**: Springer Series in Information Sciences, Springer–Verlag, Berlin, 2001).

[59] P. Koiran, Dynamics of discrete time, continuous state Hopfield networks, *Neural Computation* **6** (3) (1994) 459–468.

[60] P. Koiran, A family of universal recurrent networks, *Theoretical Computer Science* **168** (2) (1996) 473–480.

[61] J. Komlós and R. Patury, Convergence results in an associative memory model, *Neural Networks* **1** (3) (1988) 239–250.

[62] R.A. Legenstein and W. Maass, Foundations for a circuit complexity theory of sensory processing, in: *Advances in Neural Information Processing Systems* **13**, *NIPS 2000* (The MIT Press, 2001) to appear.

[63] M. Lepley and G. Miller, Computational power for networks of threshold devices in asynchronous environment, Technical, Department of Mathematics, MIT, 1983.

[64] J. Lipscomb. On the computational complexity of finding a connectionist model's stable state vectors, M.Sc. thesis, Department of Computer Science, University of Toronto, 1987.

[65] O.B. Lupanov, Implementing the algebra of logic functions in terms of bounded depth formulas in the basis +,*,-, *Soviet Physics Doklady* **6** (1961) 107–108.

[66] O.B. Lupanov, Circuits using threshold elements, *Soviet Physics Doklady* **17** (2) (1972) 91–93.

[67] W. Maass, On the computational complexity of networks of spiking neurons, in: G. Tesauro, D.S. Touretzky, and T.K. Leen, eds., *Advances in Neural Information Processing Systems* **7**, *NIPS'94* (The MIT Press, Cambridge, 1995) 183–190.

[68] W. Maass, On the computational power of noisy spiking neurons, in: D. Touretzky, M.C. Mozer, and M.E. Hasselmo, eds., *Advances in Neural Information Processing Systems* **8**, *NIPS'95* (The MIT Press, Cambridge, 1996) 211–217.

[69] W. Maass, Lower bounds for the computational power of networks of spiking neurons, *Neural Computation* **8** (1) (1996) 1–40.

[70] W. Maass, Bounds for the computational power and learning complexity of analog neural nets, *SIAM Journal on Computing* **26** (3) (1997) 708–732.

[71] W. Maass, Fast sigmoidal networks via spiking neurons, *Neural Computation* **9** (2) (1997) 279–304.

[72] W. Maass, Networks of spiking neurons: the third generation of neural network models, *Neural Networks* **10** (9) (1997) 1659–1671.

[73] W. Maass, On the Computational Power of Winner–Take–All, *Neural Computation* **12** (11) (2000) 2519–2536.

[74] W. Maass and C.M. Bishop, eds., *Pulsed Neural Networks* (The MIT Press, Cambridge, MA, 1998).

[75] W. Maass and T. Natschläger, Networks of spiking neurons can emulate arbitrary Hopfield nets in temporal coding, *Network: Computation in Neural Systems* **8** (4) (1997) 355–371.

[76] W. Maass and T. Natschläger, A model for fast analog computation based on unreliable synapses, *Neural Computation* **12** (7) (2000) 1679–1704.

[77] W. Maass and P. Orponen, On the effect of analog noise in discrete-time analog computations, *Neural Computation* **10** (5) (1998) 1071–1095.

[78] W. Maass and B. Ruf, On computation with pulses, *Information and Computation* **148** (2) (1999) 202–218.

[79] W. Maass, G. Schnitger, and E.D. Sontag, On the computational power of sigmoid versus Boolean threshold circuits, in: *Proceedings 32nd Annual Symposium on Foundations of Computer Science FOCS'91, San Juan, Puerto Rico* (IEEE Press, New York, 1991) 767–776.

[80] W. Maass and E.D. Sontag, Analog neural nets with Gaussian or other common noise distribution cannot recognize arbitrary regular languages, *Neural Computation* **11** (3) (1999) 771–782.

[81] S. Mahajan and H. Ramesh, Derandomizing approximation algorithms based on semidefinite programming, *SIAM Journal on Computing* **28** (5) (1999) 1641–1663.

[82] 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** (4) (1987) 461–482.

[83] M.L. Minsky and S.A. Papert, *Perceptrons* (The MIT Press, Cambridge, MA, 1969).

[84] S. Muroga, *Threshold Logic and its Applications* (Wiley–Interscience, New York, 1971).

[85] S. Muroga, I. Toda, and S. Takasu, Theory of majority decision elements, *Journal of the Franklin Institute* **271** (1961) 376–418.

[86] E.I. Nechiporuk, The synthesis of networks from threshold elements, *Problemy Kibernetiki* **11** (1964) 49–62.

[87] P.E. O'Neil, Hyperplane cuts of an n-cube, *Discrete Mathematics* **1** (2) (1971) 193–195.

[88] P. Orponen, Computational complexity of neural networks: A survey, *Nordic Journal of Computing* **1** (1) (1994) 94–110.

[89] P. Orponen, The computational power of discrete Hopfield nets with hidden units, *Neural Computation* **8** (2) (1996) 403–415.

[90] P. Orponen, Computing with truly asynchronous threshold logic networks, *Theoretical Computer Science* **174** (1-2) (1997) 123–136.

[91] P. Orponen, The computational power of continuous time neural networks, in: *Proceedings of the 24th Seminar on Current Trends in Theory and Practice of Informatics SOFSEM'97, Milovy, Czech Republic* (LNCS **1338**, Springer–Verlag, Berlin, 1997) 86–103.

[92] P. Orponen, An overview of the computational power of recurrent neural networks, in: H. Hyötyniemi, ed., *Proceedings of the 9th Finnish AI Conference STeP 2000–Millennium of AI, Espoo, Finland* (Vol. **3**: "AI of Tomorrow": Symposium on Theory, Finnish AI Society, Vaasa, Finland, 2000) 89–96.

[93] I. Parberry, A primer on the complexity theory of neural networks, in: R.B. Banerji, ed., *Formal Techniques in Artificial Intelligence: A Sourcebook* (Vol. **6**: Studies in Computer Science and Artificial Intelligence, Elsevier, North–Holland, Amsterdam, 1990), 217–268.

[94] I. Parberry, *Circuit Complexity and Neural Networks* (The MIT Press, Cambridge, MA, 1994).

[95] I. Parberry and G. Schnitger, Relating Boltzmann machines to conventional models of computation, *Neural Networks* **2** (1) (1989) 56–67.

[96] S. Poljak and M. Sůra, On periodical behaviour in societies with symmetric influences, *Combinatorica* **3** (1) (1983) 119–121.

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

[98] M.J.D. Powell, Radial basis functions for multivariable interpolation: A review, in: J.C. Mason and M.G. Cox, eds., *Proceedings of the IMA Conference on Algorithms for the Approximation of Functions and Data* (Oxford Science Publications, 1985) 143–167.

[99] M. Rabin, Probabilistic automata, *Information and Control* **6** (3) (1963) 230–245.

[100] A.A. Razborov, On small depth threshold circuits, in: O. Nurmi and E. Ukkonen, eds., *Proceedings of the 3rd SWAT Scandinavian Workshop on Algorithm Theory, Helsinki, Finland* (LNCS **621**, Springer–Verlag, Berlin, 1992) 42–52.

[101] J.H. Reif and S.R. Tate, On threshold circuits and polynomial computations, *SIAM Journal on Computing* **21** (5) (1992) 896–908.

[102] F. Rosenblatt, The perceptron: A probabilistic model for information storage and organization in the brain, *Psychological Review* **65** (6) (1958) 386–408.

[103] V.P. Roychowdhury, K.-Y. Siu, and A. Orlitsky, eds., *Theoretical Advances in Neural Computation and Learning* (Kluwer Academic Publishers, Boston, 1994).

[104] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, Learning internal representations by error propagation, in: D.E. Rumelhart and D.E. McClelland, eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, (The MIT Press, Cambridge, MA, 1986) **I**, 318–362.

[105] J.E. Savage, Computational work and time on finite machines, *Journal of the ACM* **19** (4) (1972) 660–674.

[106] A.A. Schäffer and M. Yannakakis, Simple local search problems that are hard to solve, *SIAM Journal on Computing* **20** (1) (1991) 56–87.

[107] H.T. Siegelmann, Recurrent neural networks and finite automata, *Journal of Computational Intelligence* **12** (4) (1996) 567–574.

[108] H.T. Siegelmann, *Neural Networks and Analog Computation: Beyond the Turing Limit* (Birkhäuser, Boston, 1999).

[109] H.T. Siegelmann, Stochastic analog networks and computational complexity, *Journal of Complexity*, **15** (4) (1999) 451–475.

[110] H.T. Siegelmann, A. Roitershtein, and A. Ben-Hur, Noisy Neural Networks and Generalizations, in: S.A. Solla, T.K. Leen, and K.-R. Müller, eds., *Advances in Neural Information Processing Systems* **12***, NIPS'99* (The MIT Press, 2000) 335–341.

[111] H.T. Siegelmann and E.D. Sontag, Analog computation via neural networks, *Theoretical Computer Science* **131** (2) (1994) 331–360.

[112] H.T. Siegelmann and E.D. Sontag, Computational power of neural networks, *Journal of Computer System Science*, **50** (1) (1995) 132–150.

[113] J. Šíma, Hopfield languages, in: *Proceedings of the 22nd Seminar on Current Trends in Theory and Practice of Informatics SOFSEM'95, Milovy, Czech Republic* (LNCS **1012**, Springer–Verlag, Berlin, 1995) 461–468.

[114] J. Šíma, Analog stable simulation of discrete neural networks, *Neural Network World* **7** (6) (1997) 679–686.

[115] J. Šíma and P. Orponen, A continuous-time Hopfield net simulation of discrete neural networks, in: *Proceedings of the 2nd International ICSC Symposium on Neural Computation NC'2000, Berlin, Germany* (ICSC Academic Press, Wetaskiwin, Canada, 2000) 36–42.

[116] J. Šíma, P. Orponen, and T. Antti-Poika, On the computational complexity of binary and analog symmetric Hopfield nets, *Neural Computation* **12** (12) (2000) 2965–2989.

[117] J. Šíma, J. and J. Wiedermann, Theory of neuromata, *Journal of the ACM* **45** (1) (1998) 155–178.

[118] K.-Y. Siu, J. Bruck, T. Kailath, and T. Hofmeister, Depth efficient neural networks for division and related problems, *IEEE Transactions on Information Theory* **39** (3) (1993) 946–956.

[119] K.-Y. Siu and V.P. Roychowdhury, On optimal depth threshold circuits for multiplication and related problems, *SIAM Journal on Discrete Mathematics* **7** (2) (1994) 284–292.

[120] K.-Y. Siu, V.P. Roychowdhury, and T. Kailath, Depth-size tradeoffs for neural computation, *IEEE Transactions on Computers* **40** (12) (1991) 1402–1412.

[121] K.-Y. Siu, V.P. Roychowdhury, and T. Kailath, Computing with almost optimal size threshold circuits, in: *Proceedings of the IEEE International Symposium on Information Theory, Budapest, Hungary* (1991).

[122] K.-Y. Siu, V.P. Roychowdhury, and T. Kailath, Rational approximation techniques for analysis of neural networks, *IEEE Transactions on Information Theory* **40** (2) (1994) 455–466.

[123] K.-Y. Siu, V.P. Roychowdhury, and T. Kailath, *Discrete Neural Computation: A Theoretical Foundation* (Prentice Hall, Englewood Cliffs, NJ, 1995).

[124] K.-Y. Siu, V.P. Roychowdhury, and T. Kailath, Toward massively parallel design of multipliers, *Journal of Parallel and Distributed Computing* **24** (1) (1995) 86–93.

[125] M. Šorel and J. Šíma, Robust implementation of finite automata by recurrent RBF networks, in: *Proceedings of the 27th Seminar on Current Trends in Theory and Practice of Informatics SOFSEM'2000, Milovy, Czech Republic* (LNCS **1963**, Springer–Verlag, Berlin, 2000) 431–439.

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

[127] M. Tchuente, Sequential simulation of parallel iterations and applications. *Theoretical Computer Science* **48** (2-3) (1986) 135–144.

[128] J. von Neumann, Probabilistic logics and the synthesis of reliable organisms from unreliable components, in: C.E. Shannon and J. McCarthy, eds., *Automata Studies* (Vol. **34** of Annals of Mathematics Studies, Princeton University Press, NJ, 1956) 43–98.

[129] I. Wegener, *The Complexity of Boolean Functions* (Wiley/Teubner, Chichester, 1987).

[130] I. Wegener, Optimal lower bounds on the depth of polynomial-size threshold circuits for some arithmetics functions, *Information Processing Letters* **46** (2) (1993) 85–87.

[131] J. Wiedermann, Complexity issues in discrete neurocomputing, *Neural Network World* **4** (1) (1994) 99–119.

[132] Y.A. Zuev, Asymptotics of the logarithm of the number of threshold functions of the algebra of logic, *Soviet Mathematics Doklady* **39** (3) (1989) 512–513.