General Purpose Computation with Neural Networks: A Survey of Complexity Theoretic Results

Jiří Šíma*
Institute of Computer Science,
Academy of Sciences of the Czech Republic,
P. O. Box 5, 18207 Prague 8, Czech Republic, *sima@cs.cas.cz*

Pekka Orponen†
Laboratory for Theoretical Computer Science,
Helsinki University of Technology,
P. O. Box 5400, FIN-02015 HUT, Finland, *orponen@tcs.hut.fi*

Abstract

We survey and summarize the existing literature on the computational aspects of neural network models, by presenting a detailed taxonomy of the various models according to their complexity theoretic characteristics. The criteria of classification include e.g. the architecture of the network (feedforward vs. recurrent), time model (discrete vs. continuous), state type (binary vs. analog), weight constraints (symmetric vs. asymmetric), network size (finite nets vs. infinite families), computation type (deterministic vs. probabilistic), etc. The underlying results concerning the computational power and complexity issues of perceptron, RBF, winner-take-all, and spiking neural networks are briefly surveyed, with pointers to the relevant literature. In our survey we focus mainly on the digital computation whose inputs and outputs are binary in nature, although their values are quite often encoded as analog neuron states. Also we completely omit the important learning issues.

*Research partially supported by grants GA AS CR B2030007, GA CR No. 201/02/1456.

†Research partially supported by grant 81120 from the Academy of Finland.

1 Introduction

1.1 Computational Complexity of Neural Networks

Neural networks are being investigated in many fields of artificial intelligence as a computational paradigm alternative to the conventional von Neumann model. The computational potential and limits of conventional computers are by now well understood in terms of classical models of computation such as Turing machines. Analogously, many fundamental theoretical results have been achieved in the past decade concerning the capabilities of neural networks for general computation (Hartley & Szu, 1987; Muroga, 1971; Orponen, 1994; Orponen, 2000; Parberry, 1990; Parberry 1994; Roychowdhury, Siu, & Orlitsky, 1994; Siegelmann 1999a; Šíma, 2001; Siu, Roychowdhury, & Kailath, 1995a, Wiedermann, 1994).

In particular, the computational power of neural nets has been investigated by comparing their various models with each other and with more traditional computational models and descriptive tools such as finite automata, regular expressions, grammars, Turing machines, Boolean circuits, etc. The aim of this approach is to find out what is, either ultimately or efficiently, computable by particular neural models, and how to optimally implement the required functions. The main technical tool of this investigation is computational complexity theory (Balcázar, Díaz, & Gabarró, 1995; Bovet & Crescenzi, 1993; Papadimitriou, 1994). As a result, a complexity theoretic taxonomy of neural networks has evolved, enriching the traditional repertoire of formal computational models and even pointing out new sources of efficient computation.

In addition, since neural network models have also been inspired by neurophysiological knowledge, the respective theoretical results may help to broaden our understanding of the computational principles of mental processes.

1.2 A Formal Computational Model of Neural Network

Let us first recall a general model of an (artificial) neural network that 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 we assume that 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 representing the *architecture* of the network, in which also each edge (i, j) leading from neuron i to j is labeled with a real *(synaptic) weight* $w(i, j) = w_{ji} \in \mathbf{R}$. 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*) $y_j^{(t)} \in \mathbf{R}$ as a function of time $t \geq 0$. This establishes the *network state* $\mathbf{y}^{(t)} = (y_1^{(t)}, \ldots, y_s^{(t)}) \in \mathbf{R}^s$ at each time instant $t \geq 0$. At the beginning of a 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 by a selected subset of neurons collecting their *inputs* from the outputs of their incident neurons via the underlying weighted connections, and transforming these input values into their current states. Finally, a global output from the network is read at the end or even in the course of computation.

In the present survey, we mainly consider *digital* computations over *binary* external inputs providing *binary* outputs, although these values may be encoded as analog neuron states, and the intermediate stages of computation often operate with real numbers.

1.3 An Outline of the Survey

Neural network models can be classified according to the restrictions that are imposed on their parameters and/or computational dynamics. In this way various model classes are obtained which have different computational capabilities. In the present paper a detailed taxonomy of such classes is presented from the complexity theoretic point of view. The respective theoretical results concerning the computational power and complexity issues of neural networks are briefly surveyed and complemented by relevant references. Since the presented results share computational complexity theory as their main technical tool, we recall in an appendix to the paper the basic definitions of the key complexity classes. On the other hand, we completely omit the pertinent learning issues whose complexity aspects would deserve a separate survey.

The structure of this survey partially follows the underlying taxonomy of neural network models as depicted in Figure 1 where the numbers of corresponding sections are indicated in parentheses. In section 2 we focus on the classical *perceptron networks*, whose computational aspects are by now well understood. Both their *discrete-* (sections 2.1—2.4) and *continuous-time* (section 2.5) dynamics are considered. Besides a single perceptron (section 2.1), the models of discrete-time perceptron networks are basically divided according to their architectures into *feedforward* (section 2.2) and *recurrent* (section 2.3) networks corresponding to bounded and unbounded computations, respectively. This architectural criterion is also applied in the classification of *probabilistic* perceptron networks in section 2.4. The computational taxonomy of discrete-time perceptron networks is further refined into *binary* (e.g. section 2.2.1) and *analog* networks (e.g. section 2.2.2), according to the permissible domains of neuron state values. Also *finite* networks (e.g. section 2.3.1) and *infinite families* of networks (e.g. section 2.3.4) are distinguished. In addition, the computational properties of *symmetric* recurrent networks (section 2.3.3), i.e. networks with undirected architecture graphs, differ from those of general *asymmetric* ones (section 2.3.2).

In section 3 we proceed to survey other neural network models, whose computational capabilities have been studied only recently. This includes the *radial basis function* (section 3.1) and *winner-take-all* (section 3.2) networks which employ

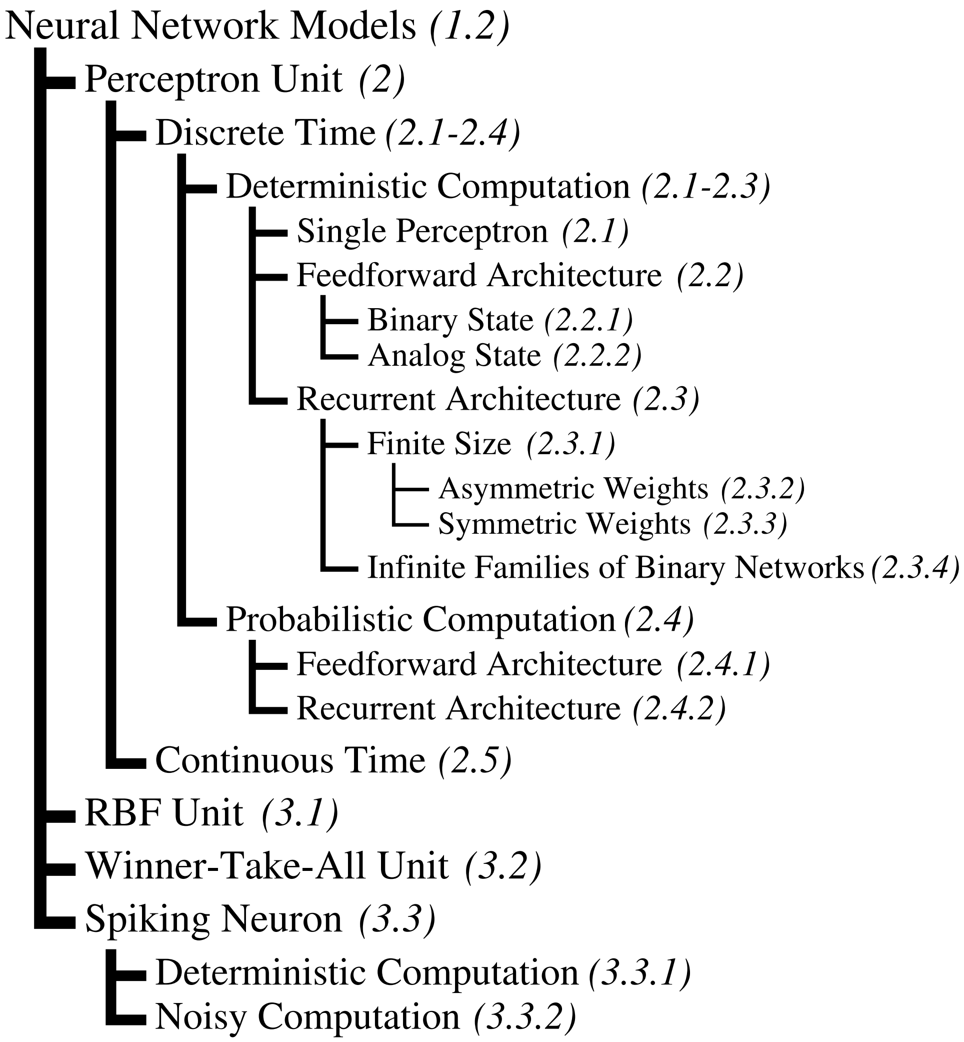

Figure 1: A Taxonomy of Neural Network Models.

computational units that are alternative to the perceptrons. Finally, 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* (section 3.3.1) and *noisy* (section 3.3.2) spiking networks are reviewed.

We conclude with overall comparison of the results in section 4 where the main open problem areas are outlined. A preliminary version of this paper appeared as an extended abstract (Šíma, 2001).

2 Perceptron Networks

The *perceptron* networks represent the most popular neural network model. Following current practice, the term perceptron is here applied in a more general way than in the book by Rosenblatt (1958) and covers also the types of units that were later derived from the original perceptron (e.g. sigmoidal gates). In this section, the computational aspects of perceptron networks will be inspected mainly for discrete time.

A 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 its inputs. This includes a *bias* value $w_{j0} \in \mathbf{R}$, which can be viewed as the weight from a formal constant unit input $y_0^{(t)} \equiv 1$. Recall from section 1.2 that some of the weights w_{ji} $(1 \leq i \leq s)$ in equation 2.1 may be zero indicating the absence of a connection from neuron i to neuron j.

At the next instant $t+1$, the neurons $j \in \alpha_{t+1}$ from a selected subset $\alpha_{t+1} \subseteq V$ compute their new outputs $y_j^{(t+1)}$ by applying an *activation function* $\sigma : \mathbf{R} \longrightarrow \mathbf{R}$ 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.

Perceptron networks with *binary states* $y_j \in \{0, 1\}$ (shortly *binary networks*) usually employ the *hard limiter* or *Heaviside* activation function

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

and the corresponding units are often called *threshold gates*. 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 (Parberry, 1990; Parberry, 1994).

Analog-state networks (shortly *analog networks*), on the other hand, approximate the Heaviside function defined in equation 2.3 with some continuous sigmoidal 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*

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

and the corresponding units are also called the *sigmoidal gates*. Hence the outputs from sigmoidal gates 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 $h \in \mathbf{R}$.

Now, the computational aspects of discrete-time perceptron networks will be surveyed. We start with a single perceptron unit in section 2.1. Then we will focus on the feedforward and recurrent architectures in section 2.2 and 2.3, respectively. Furthermore, we also consider probabilistic computation with discrete-time perceptron networks in section 2.4. Finally, the deterministic model of perceptron networks will be extended to continuous time in section 2.5.

2.1 Single Perceptron

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 Boolean functions can be implemented by a single perceptron, the parity (XOR) function being the most prominent example requiring more complicated devices (Minsky & Papert, 1969). Boolean functions that are computable by perceptrons are also called *linear threshold functions*. This important function class 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 Boolean threshold functions over n variables is approximately $2^{\Theta(n^2)}$ which is a very small fraction of all the 2^{2^n} Boolean functions. The upper bound $2^{n^2 - n\log_2 n + O(n)}$ was already given by Schläfli (1901; see also Cover, 1965) whereas the almost matching lower bound $2^{n^2 - n\log_2 n - O(n)}$ has only recently been achieved (Kahn, Komlós, & Szemerédi, 1995). These bounds have further been improved by Irmatov (1996). Moreover, it is known to be a co-NP-complete problem to decide whether a Boolean function given in disjunctive or conjunctive normal form is a linear threshold function (Hegedüs & Megiddo, 1996).

Also the descriptive complexity of linear threshold functions has been studied. It is known that any Boolean linear threshold function can be implemented by using only *integer* weights (Minsky & Papert, 1969), and that the number of bits that are necessary (Håstad, 1994) and sufficient (Muroga, Toda, & Takasu, 1961) for representing a single integer weight parameter in an n-input perceptron is $\Theta(n \log n)$.

2.2 Feedforward Networks

The architecture of a *feedforward* (*multi-layered*) neural network is an acyclic graph. Hence, units in a feedforward network can be grouped in a unique minimal way 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, or *input layer* α_0 consists of n external inputs and is not counted in the number of layers and in the network size. The last, or *output layer* α_d is composed of m output neurons. The hidden units are contained in the intermediate *hidden layers* $\alpha_1, \ldots, \alpha_{d-1}$. The number of layers d excluding the input one is called the network *depth* (e.g. two-layered networks have depth 2). Computation proceeds layer by layer from the input layer via the hidden layers up to the output layer. Initially, the states of the input units are set to represent an external input. In a general step, supposing all the outputs of neurons have been determined up to a certain layer $t < d$, the states of neurons in the next layer α_{t+1} are computed in parallel according to formulas 2.1 and 2.2. In 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 within a parallel time equal to the depth d of the network.

The above-described model of feedforward neural networks coincides with the model of *Boolean threshold circuits* (hereafter briefly *circuits*), where the computational units are more commonly called (threshold) gates. The computational aspects of circuits have been intensively studied in the area of Boolean circuit complexity (Savage, 1998; Vollmer, 1999; Wegener, 1987). A single circuit C with n inputs 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 for circuits such as size $S(n)$, depth $D(n)$, maximum weight $W(n)$, etc. (see below) are expressed in terms of the input length n. For example, in a *polynomial-size* family of circuits, the function $S(n)$ is bounded by a polynomial $O(n^c)$ (for some constant c), in a *constant-depth* family the circuits have constant $D(n) = c$, etc.

By the above definition, infinite families of circuits can in general be *nonuniform* in the sense of not possessing any finite description relating the circuits for different input lengths to each other. Hence such families can compute functions that are not computable by any finitely described algorithm (Turing machine), and thus possess "super-Turing" computational capabilities trivially by virtue of their definition. To avoid such pathologies, one often considers *uniform* circuit families, where it is required that a fixed Turing machine exists that for each input length $n \geq 0$ constructs the appropriate circuit C_n from this family. Uniform circuit families are computationally equivalent to Turing machines.

2.2.1 Binary-State Feedforward Networks

In this section the computational capabilities of binary-state feedforward neural networks that employ the Heaviside activation function (equation 2.3) will be inspected.

Computational Universality: The simplest case of this model—a single perceptron—has already been considered in section 2.1, and it is clear that threshold gates must be connected into bigger circuits to be able to implement arbitrary Boolean functions. In this setting, for any vector Boolean function $\mathbf{f} : \{0,1\}^n \longrightarrow \{0,1\}^m$, a threshold circuit can be constructed using

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

gates and depth 4 (Lupanov, 1972). This construction is optimal, as there are functions that provably require a matching number of gates even for unbounded depth (Horne & Hush, 1994). The respective lower (Nechiporuk, 1964) and upper (Lupanov, 1961) bounds had actually been earlier estimated for single-output networks and coincide with the size given in equation 2.6 for $m = 1$. In addition, Cover (1968) derived a corresponding lower bound for the number of connections required in threshold circuits computing arbitrary Boolean functions which can be expressed as formula 2.6 squared. These results imply that (nonuniform) infinite families of constant-depth threshold circuits with exponentially many gates are capable of computing any input-output mapping in constant parallel time.

Polynomial Weights: In 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, such polynomial weights reduce the computational power of threshold circuits since from section 2.1 we know that 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 of a circuit by at most one layer and imposing a polynomial overhead in its size (Goldmann, Håstad, & Razborov, 1992). Goldmann and Karpinski (1998) proposed an explicit construction of the underlying polynomial-weight circuit whose polynomial size-increase is independent of the depth. Thus, polynomial weights can be assumed in multi-layered perceptron networks if one extra computational step can be granted.

Another input convention assumes that each input Boolean variable is presented to the circuit together with its negation. In this case any threshold circuit can be modified to contain only *positive* weights, while the size of the circuit is at most doubled, and its depth and size of weights stay the same (Hajnal, Maass, Pudlák, Szegedy, & Turán, 1993).

Bounded Fan-In: Another restriction concerns the maximum *fan-in* of gates in the circuit, i.e. the maximum number of inputs to a single unit. Circuits with bounded fan-in represent a model of conventional circuit technology in which the number of input wires to each gate is restricted. Unbounded fan-in is a typical property of densely interconnected neural networks, which thereby may gain more efficiency. For example, any circuit of size s, depth d, and with maximum fan-in 2 can be implemented by a threshold circuit with unbounded fan-in and $O(s^2)$ gates whose depth is $O(d/\log\log s)$ (Chandra, Stockmeyer, & Vishkin, 1984). This yields a speed-up factor of $O(\log\log n)$ in polynomial-size feedforward neural networks.

Polynomial Size and Constant Depth: The exponential size for the universal networks given in equation 2.6 is not practically realizable for large input lengths. Actually, no naturally interesting function (e.g. from the complexity class NP) has so far been decisively proved to require even a superlinear number of gates in circuits of unbounded depth, but intuitively functions requiring large circuits abound (Wegener, 1987). Nevertheless, many important functions can be computed by threshold circuits of polynomial size and constant depth as will be seen below. In addition to their implementation interest, such results are also consonant with the neurophysiological data indicating that quite complicated functions are computed using only a few layers of brain structure.

It is conventional to denote by TC^0_d, $d \geq 1$, the class of all functions computable by polynomial-size and polynomial-weight threshold circuits of depth d. (An alternative notation is $\widehat{LT}_d$, where the unadorned notation LT_d refers to the corresponding unbounded-weight classes.) The complexity class

$$TC^0 = \bigcup_{d\geq 1} TC^0_d \tag{2.7}$$

then encompasses all the functions computable in constant parallel time by such networks. Obviously,

$$TC^0_1 \subseteq TC^0_2 \subseteq TC^0_3 \subseteq \cdots \tag{2.8}$$

is a possible TC^0 *hierarchy* that is depicted in Figure 2 together with the corresponding probabilistic complexity classes RTC^0_k discussed below in section 2.4.1. For example, class TC^0_1 which contains all the functions computable by single perceptrons with polynomial weights is certainly a proper subclass of TC^0_2, i.e.

$$TC^0_1 \subsetneqq TC^0_2\,, \tag{2.9}$$

since e.g. the parity (XOR) function is not computable by a single perceptron, but is easily computed by a small depth-2 circuit.

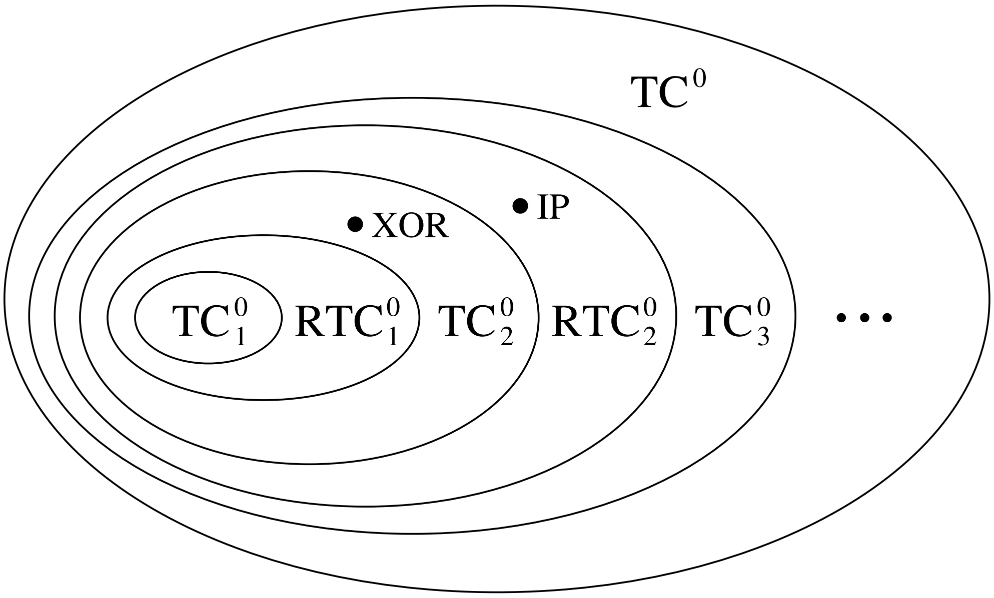

Figure 2: A possible TC^0-hierarchy with small depth separations.

A less known result is the deeper separation

$$TC_2^0 \subsetneqq TC_3^0 \tag{2.10}$$

(Hajnal, Maass, Pudlák, Szegedy, & Turán, 1993) which is also of great importance in practical applications of feedforward networks where one usually decides between the use of two- or three-layered architectures. The witnessing example of a function that belongs to $TC_3^0 \setminus TC_2^0$ is the *Boolean inner product* $IP : \{0,1\}^{2k} \longrightarrow \{0,1\}$ $(k \geq 1)$ defined as

$$IP\,(x_1, \ldots, x_k, x_1', \ldots, x_k') = \bigoplus_{i=1}^{k} AND\,(x_i, x_i') \tag{2.11}$$

where $\bigoplus$ stands for the k-bit parity function that has result 1 whenever the number of 1's among k input bits is odd. This result means that polynomial-size and polynomial-weight three-layered perceptron networks are computationally more powerful than two-layered ones. However, the corresponding separation for large-weight networks is still an open problem.

The separation of the TC^0 hierarchy above depth 3 is unknown: this represents one of the most important open problems in contemporary complexity theory. It is at this moment conceivable that all problems in the complexity class P (or even NP) could be solved by depth-3 threshold circuits with a linear number of gates. In this case the TC^0 hierarchy would collapse at depth 3, i.e. $TC^0 \subseteq TC_3^0$.

Symmetric Boolean Functions: An important class of functions are the *symmetric* Boolean functions whose values are independent of the order of input variables, i.e. the value of a symmetric function depends only on the number of 1's in the input. A typical example is the parity function. It is known that any symmetric function over n variables can be implemented by a polynomial-weight threshold circuit of size $O(\sqrt{n})$ gates and depth 3 (Siu, Roychowdhury, & Kailath, 1991). 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; for depth 2 O'Neil (1971) achieved even a matching size lower bound $\Omega(\sqrt{n})$.

This also illustrates the fact that threshold circuits are computationally more powerful than so-called *AND-OR circuits* that employ only a basis of logic gates consisting of the AND, OR, and NOT functions. In particular, Furst, Saxe, and Sipser (1984) proved that the parity function cannot be computed by polynomial-size constant-depth AND-OR circuits (corresponding to a complexity class denoted AC^0); a stronger exponential-size lower bound was shown by Yao (1985) and later simplified by Håstad (1989). This has led to a conjecture of a weaker TC^0 hierarchy collapse, such that $AC^0 \subseteq TC^0_3$. A partial result along this line is that AC^0 functions are computable by threshold circuits of depth 3 and subexponential size $n^{\log^c n}$, for some constant c (Allender, 1989).

Arithmetic Functions: The computational power of polynomial-size and polynomial-weight threshold circuits of small constant depth can further be illustrated by their capability of computing the basic arithmetic functions (Razborov 1992; Roychowdhury, Siu, & Orlitsky, 1994; Siu, Roychowdhury, & Kailath, 1995a). For example, two-layered perceptron networks of polynomial size and weights can be constructed for comparing and computing the sum of two n-bit binary numbers (Alon & Bruck, 1994) or even multiple sum of n such numbers, as follows from the work by Siu and Roychowdhury (1994) by using the result due to Goldmann and Karpinski (1998). Further, the product and quotient of two n-bit binary numbers, powering to n (Siu & Roychowdhury, 1994), and sorting (Siu, Bruck, Kailath, & Hofmeister, 1993) of n such numbers can be achieved with three-layered feedforward networks. It is also known that the depth in these circuits (except for powering where this issue is still unresolved) cannot be reduced when polynomial size and weights are required: generally this follows from the result due to Hajnal, Maass, Pudlák, Szegedy, and Turán (1993). More explicit proofs for division were presented by Hofmeister, Pudlák (1992), and Wegener (1993), while Siu, Bruck, Kailath, and Hofmeister (1993) showed the argument for sorting. Moreover, the multiple product of n n-bit binary numbers is performable in depth 4 (Siu & Roychowdhury, 1994). These depth complexity results for polynomial-size and polynomial-weight threshold circuits implementing the basic arithmetic functions are summarized in Table 1 that was first presented by Razborov (1992) and later

Function	Lower bound	Upper bound
Comparison	2	2
Addition	2	2
Multiple Addition	2	2
Multiplication	3	3
Division	3	3
Powering	2	3
Sorting	3	3
Multiple Multiplication	3	4

Table 1: The number of layers in feedforward networks of polynomial size and weights computing arithmetic functions.

updated by Hofmeister (1994). It thus follows that any analytic function can be approximated to high precision by a perceptron network of polynomial size and weights, and using only a small constant number of layers, given as input an n-bit binary representation of its real argument (Reif & Tate, 1992).

Trade-Off Results: 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 with polynomial weights for some of the arithmetic tasks discussed above if the permissible depth is increased by a few layers, e.g. a two-number multiplier implementation with $O(n^2)$ threshold gates in depth 4 (Hofmeister, Hohberg, & Köhling, 1991) which reduces the size $O(n^{27} \log n)$ of the corresponding optimal-depth implementation by three-layered networks (Siu & Roychowdhury, 1994).

Further, general trade-off results are known for the parity function (Siu, Roychowdhury, & Kailath, 1993), showing that 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. The corresponding lower bound on the size of any depth-$(d+1)$, polynomial-weight threshold circuit computing the parity function is $\Omega(n^{1/d}/\log^2 n)$. Another example of trade-offs between the depth and the maximum fan-in is an implementation of any n-variable symmetric Boolean function 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 $d > 0$ (Siu, Roychowdhury, & Kailath, 1995b).

Total Wire Length: Recently, so called total wire length has been introduced (Legenstein & Maass, 2001) as a new complexity measure for feedforward networks. This measure tries to take into account some significant VLSI implementation issues in, e.g. networks for efficient sensory processing where the number of parallel inputs is quite large. One possible model here assumes that the gates are placed at the intersection points of a 2-dimensional grid, with unit distance between adjacent intersection points. The gates can be arbitrarily connected in the plane by wires which may cross. The quantity of interest is then the sum of wire lengths in the circuit, and the minimal value of this taken over all possible placements of the gates defines the *total wire length* of the circuit. General k-input (threshold) gates are here modeled as microcircuits with unit evaluation time, each occupying a set of k intersection points of the grid and 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 $\Omega(n^{2.5})$. For example, a Boolean function $P_{LR}^k : \{0,1\}^{2k} \longrightarrow \{0,1\}$ ($k \geq 2$) defined as

$$P_{LR}^k(x_1,\ldots,x_k,x_1',\ldots,x_k') = 1 \quad \text{iff} \quad (\exists\, 1 \leq i < j \leq k)\;\; x_i = x_j' = 1\,, \tag{2.12}$$

representing a simple global pattern detection prototype, can be computed by a two-layered network with $2\log_2 k + 1$ threshold gates and total wire length $O(k \log k)$ (Legenstein & Maass, 2001).

2.2.2 Analog-State Feedforward Networks

The computational capabilities of binary-state feedforward neural networks will now be compared to those of analog units employing e.g. the standard sigmoid activation function defined in equation 2.5 (although in fact the following results are valid for much more general classes of sigmoidal gates). This is probably the most common computational model used in practical neural networks, e.g. in back-propagation learning (Rumelhart, Hinton, & Williams, 1986; Werbos, 1974). Obviously, Boolean threshold circuits can be implemented by analog feedforward networks of the same architecture by increasing the absolute values of weights when a more precise approximation of the function defined in equation 2.3 by the one defined in formula 2.5 is needed (Maass, Schnitger, & Sontag, 1991).

Constant Size: On the other hand, it appears that using real-valued states 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

$$\begin{aligned} &F_k(x_1,\ldots,x_k,x_1',\ldots,x_k') \\ &\qquad = Majority(x_1,\ldots,x_k) \oplus Majority(x_1',\ldots,x_k') \end{aligned} \tag{2.13}$$

where

$$Majority(x_1,\ldots,x_k) = 1 \quad \text{iff} \quad \sum_{i=1}^{k} x_i \geq \frac{k}{2}\,, \tag{2.14}$$

and $\oplus$ stands for the XOR function, can be computed by two-layered networks having only five analog units and bounded 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 a constant number of binary-output gates, even with unbounded weights (Maass, Schnitger, & Sontag, 1991). This confirms that analog feedforward networks are somewhat more powerful computational devices than their binary counterparts.

A similar but *depth-independent* result was shown for the *unary squaring functions* $SQ_k : \{0,1\}^{k^2+k} \longrightarrow \{0,1\}$ ($k \geq 1$), defined as

$$SQ_k(x_1, \ldots, x_k, z_1, \ldots, z_{k^2}) = 1 \quad \text{iff} \quad \left(\sum_{i=1}^{k} x_i \right)^2 \geq \sum_{i=1}^{k^2} z_i \,. \tag{2.15}$$

These functions 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 (DasGupta & Schnitger, 1996). Thus, the size of feedforward networks can sometimes be reduced by a logarithmic factor when threshold gates are replaced by sigmoidal ones.

Polynomial Size: The complexity classes TC_d^0 can be generalized to their analog versions $TC_d^0(\sigma)$ ($d \geq 1$) 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 the activation function defined in equation 2.5. It turns out that at this level of consideration, the analog and binary classes coincide, i.e.

$$TC_d^0(\sigma) = TC_d^0 \tag{2.16}$$

for every $d \geq 1$ (Maass, Schnitger, & Sontag, 1991). In other words, there is no substantial difference in the computational power of polynomial-size analog- and binary-state feedforward networks of the same fixed depth, and in particular three-layer networks maintain their computational advantage over two-layer networks even in this case. This computational equivalence of polynomial-size analog and binary networks remains valid 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 (DasGupta & Schnitger, 1993).

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 defined in equation 2.4 have been shown to belong to the class TC^0 (Maass, 1997a).

Finally, the trade-off lower bounds 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 $\delta > 0$ (Siu, Roychowdhury, & Kailath, 1994), which is almost optimal since as we already know $O(dn^{1/d})$ units are sufficient.

2.3 Recurrent Neural Networks

The architecture underlying *recurrent* (*cyclic*) neural networks is in general a cyclic graph, and hence computations on these devices may not terminate in a bounded number of steps. In this context, special attention has been paid to *symmetric* or *Hopfield* networks whose weights satisfy

$$w(i,j) = w(j,i) \tag{2.17}$$

for every $i, j \in V$, and consequently the underlying architecture is an undirected graph.

In order to specify the computational dynamics of cyclic networks completely one needs to add some conventions. For example, various computational modes are possible for recurrent networks, depending on the choice of sets α_t of updated units in equation 2.2. A recurrent network is said to operate in *sequential* mode if at every time instant $t \geq 1$ at most one unit updates its state according to formula 2.2, that is $|\alpha_t| \leq 1$. In order to formally avoid long constant intermediate computations when only those units are updated that effectively do not change their outputs, the notion of a *productive* computation of length $t^\star$ discrete updates is introduced, meaning that for every $1 \leq t \leq t^\star$ there exists a unit $j \in \alpha_t$ updated at time t such that $y_j^{(t)} \neq y_j^{(t-1)}$. 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 some of them may update their outputs even several times).

For a *parallel* mode, in contrast, there exists a time instant $t \geq 1$ when at least two neurons re-compute their new outputs simultaneously, i.e. with $|\alpha_t| \geq 2$. 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 $t \geq 1$.

In *asynchronous* computations the choice of the update set α_t is random so that each unit in the network may in fact decide independently at which time instant its state is updated. In contrast, in *synchronous* computations the sets α_t are predestined deterministically for each time instant t. The seemingly less powerful asynchronous models have been shown for binary states to have the same computational power as their systematic synchronous counterparts, 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

(Orponen, 1997b). Hence, a synchronous model will mostly be assumed in the sequel.

For the purpose of universal computations over inputs of arbitrary lengths, different input protocols have been introduced. Thus in the following both *finite* networks with a finite number of units (section 2.3.1) and infinite families of cyclic networks, one for each input length (section 2.3.4), are considered. Particularly for the finite recurrent networks we distinguish between the asymmetric and symmetric weights in section 2.3.2 and 2.3.3, respectively.

2.3.1 Finite Neural Language Acceptors

The computational power of recurrent networks has been studied analogously to the traditional models of computations so that the networks are exploited as acceptors of languages $L \subseteq \{0,1\}^\star = \cup_{n\geq 0}\{0,1\}^n$ over the binary alphabet. In this context, a language L corresponds to a decision problem in the sense that L is composed of those input instances for which the answer to the problem is "yes". For example, the problem of deciding whether a given Boolean formula is satisfiable is associated with language SAT containing all satisfiable Boolean formulas. Recall that a language $L \subseteq \{0,1\}^\star$ with characteristic function $c_L : \{0,1\}^\star \longrightarrow \{0,1\}$ (i.e. a word $\mathbf{x} \in \{0,1\}^\star$ belongs to language L iff $c_L(\mathbf{x}) = 1$) is *accepted* by a computational model M (i.e. the corresponding decision problem is solvable by M) if M partially implements c_L in the sense that M may possibly not terminate for inputs $\mathbf{x} \notin L$. Furthermore, a language L is *recognized* by M if c_L is totally implemented by M.

In the case of finite networks, or so called *neural acceptors* working under fully parallel updates, an input word (string) $\mathbf{x} = x_1 \ldots x_n \in \{0,1\}^n$ of arbitrary length $n \geq 0$ is sequentially presented to the network bit by bit via an input neuron $in \in V$. The state of this unit is externally set to the respective input bits at prescribed time instants, regardless of any influence from the remaining units in the network. The output neuron $out \in V$ subsequently signals whether the input word $\mathbf{x}$ belongs to the underlying language L.

Especially in the case of *binary-state* neural acceptors (shortly *binary neural acceptors*), the input bits $x_1, \ldots, x_n$ are sequentially presented, each only once in a given *period* of $p \geq 1$ discrete steps, that is

$$y_{in}^{(p(i-1))} = x_i \tag{2.18}$$

for $i = 1, \ldots, n$, and the output neuron recognizes each prefix of the input word online, possibly with some time delay $k \geq 1$, that is

$$y_{out}^{(p(i-1)+k+1)} = \begin{cases} 1 & \text{if } x_1 \ldots x_i \in L \\ 0 & \text{if } x_1 \ldots x_i \notin L\,. \end{cases} \tag{2.19}$$

It has been shown for binary networks that the constant time delay k can always be reduced to 1 with only a linear increase in network size (Šíma & Wiedermann, 1998).

	Power of det. networks			Power of prob. networks	
Weight Domain	unbounded time	polynomial time	Probab. Domain	unbounded time	polynomial time
integer	regular	regular	real	regular	regular
rational	recursive	P	rational	recursive	BPP
			real	arbitrary	Pref-BPP/log
real	arbitrary	P/poly	real	arbitrary	P/poly

Table 2: The computational power of deterministic and probabilistic discrete-time analog neural networks with the saturated-linear activation function.

In the case of *analog-state* networks usually $p = 1$, and an additional *validation* input unit $ival \in V$ is employed to indicate the end of an input word, i.e.

$$y_{ival}^{(t)} = \begin{cases} 1 & \text{for } t = 0, \ldots, n-1 \\ 0 & \text{for } t \geq n\,. \end{cases} \tag{2.20}$$

Correspondingly, the output neuron after some time $T(n) \geq n$ that depends on the input length n provides the result of recognition

$$y_{out}^{(t)} = \begin{cases} 1 & \text{if } \mathbf{x} \in L \text{ and } t = T(n) \\ 0 & \text{otherwise}\,, \end{cases} \tag{2.21}$$

which is again announced by a validation output unit $oval \in V$, that is

$$y_{oval}^{(t)} = \begin{cases} 1 & \text{for } t = T(n) \\ 0 & \text{for } t \neq T(n)\,. \end{cases} \tag{2.22}$$

In analog networks, an alternative input protocol is also possible (Siegelmann & Sontag, 1995), by encoding an input word $\mathbf{x}$ of arbitrary length n into a real initial state of the input neuron, e.g. as

$$y_{in}^{(0)} = \sum_{i=1}^{n} \frac{2x_i + 1}{4^i}\,. \tag{2.23}$$

2.3.2 Finite Asymmetric Recurrent Networks

The computational power of finite recurrent neural networks with in general asymmetric weights employing the saturated-linear activation function defined in equation 2.4 depends on the descriptive complexity of their weights. The respective results are summarized in Table 2 as presented by Siegelmann (1994), including the comparison with the probabilistic recurrent networks discussed below in section 2.4.2.

Description of regular language	Implementation of threshold automaton	
	Size	Period of presenting the input bits
q-state deterministic finite automaton	$\Theta(\sqrt{q})$	4
Regular expression of length ℓ	$\Theta(\ell)$	1

Table 3: The optimal size of threshold automaton implementation.

Binary-State Networks: For integer weights these models coincide with finite binary-state recurrent networks employing the Heaviside activation function defined in equation 2.3. Since binary neural acceptors composed of threshold gates have only a finite number of global network states, their computational power corresponds to that of finite automata (Kleene, 1956; cf. also Minsky, 1967), and they can be called *threshold automata* for recognizing regular languages. Finer descriptive measures have been studied (Alon, Dewdney, & Ott, 1991) to find out how efficient such neural implementations of finite automata can be. The results concerning the descriptive complexity of threshold automata are summarized in Table 3.

In particular, it has been shown that any deterministic finite automaton with q states can be simulated by a threshold automaton of size $O(\sqrt{q})$ and a constant period $p = 4$ of presenting the input bits (Horne & Hush, 1996; Indyk, 1995). Furthermore, in the worst case the size $\Omega(\sqrt{q})$ of the simulating automaton cannot be reduced if either the period is at most $p = O(\log q)$ (Horne & Hush, 1996) or polynomial weights are assumed (Indyk, 1995).

The size of threshold automata can also be compared to the length of regular expressions, which are known to have higher descriptional efficiency than deterministic finite automata. In this case, any regular language described by a regular expression of length ℓ can be recognized by a threshold automaton of optimal size $\Theta(\ell)$ (Šíma & Wiedermann, 1998). On the other hand, in some cases linear-size threshold automata can be constructed for regular languages that provably require regular expressions of exponential length, thus confirming the powerful descriptive capabilities of this model.

Analog Networks with Rational Weights: Finite analog recurrent networks with rational weights employing the saturated-linear activation function (equation 2.4) can simulate arbitrary Turing machines step by step (Siegelmann & Sontag, 1995). By implementing a universal Turing machine using this technique, it follows that 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 $O(T(n))$. The size of this universal network can further be reduced even to 25 neurons, at the cost of increasing the simulation time to $O(n^2T(n))$ (Indyk, 1995). It follows that polynomial-time computations of such finite-size networks correspond to the well-known complexity class P. In addition, Turing universality of a fixed analog network has been shown for more general classes of activation functions (Koiran, 1996) including the standard sigmoid defined in equation 2.5 (Kilian & Siegelmann, 1996), although in this case the known simulations require exponential time overhead per each computational step.

Analog Networks with Arbitrary Real Weights: Finite analog recurrent networks with arbitrary real weights employing the saturated-linear activation function (equation 2.4) can even derive "super-Turing" computational capabilities from the unbounded amount of information encodable in arbitrarily precise real numbers. 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 $T(n)$ (Siegelmann & Sontag, 1994). This means that polynomial-time computations by such networks correspond to the nonuniform complexity class P/poly. (See the appendix below or the textbook by Balcázar, Díaz, and Gabarró (1995) for a definition of this class.) Furthermore, within exponential time *any* input-output mapping can be computed by such a network with appropriately coded weights.

A proper hierarchy of nonuniform complexity classes between P and P/poly has been established for polynomial-time computations of finite analog recurrent networks with increasing Kolmogorov complexity (information content) of their real weights (Balcázar, Gavaldà, & Siegelmann, 1997). For example, setting a logarithmic bound on the resource-bounded Kolmogorov complexity of the real weights, the languages accepted correspond to the complexity class Pref-P/log, as defined in the appendix or (Balcázar & Hermo, 1998). By allowing only recursive (computable) weights the polynomial-time computations of analog networks can recognize exactly the languages from the recursive part of P/poly, which is known to be strictly stronger than P. This means that this model exhibits a speedup without the use of nonrecursive weights.

Analog Noise: All the preceding results concerning analog computations in finite recurrent networks assume arbitrary-precision real number calculations. However, subjecting the output of each analog neuron to any amount of analog noise reduces the computational power of such networks to at most that of threshold automata (Casey, 1996; Maass & Orponen, 1998). In fact, for any common noise distribution that is nonzero on a sufficiently large part of the state space such networks are even unable to recognize arbitrary regular languages (Maass & Sontag, 1999; Siegelmann, Roitershtein, & Ben-Hur, 2000). More precisely, they can recognize only so called *definite languages* (Rabin, 1963; see appendix for a definition), although with any desired reliability. On the other hand, if the noise level is bounded, then all regular languages can be recognized with perfect reliability, and the upper bounds on the size of neural acceptors presented in Table 3 remain valid for a general class of activation functions σ required only to have finite and different limits at plus and minus infinity:

$$\lim_{\xi \to -\infty} \sigma(\xi) = a \neq b = \lim_{\xi \to \infty} \sigma(\xi) \qquad \text{where} \qquad |a|, |b| < \infty \tag{2.24}$$

(Maass & Orponen, 1998; Siegelmann, 1996; Šíma, 1997).

Halting Problem: The complexity of various decision problems related to finite recurrent networks has also been studied. For example, the problem of deciding whether there exists a stable state in a given binary-state network is known to be NP-complete (Alon, 1985; Godbeer, Lipscomb, & Luby, 1988; Lipscomb, 1987; Porat, 1989). Furthermore, the halting problem of deciding whether a recurrent network terminates its computation over a given input has been shown to be PSPACE-complete for binary networks (Floréen & Orponen, 1994; Lepley & Miller, 1983; Porat, 1989). The problem is fully algorithmically undecidable for analog networks with rational weights and only 25 units, as follows from the computational universality of such networks (Indyk, 1995). 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 $t^\star$ (Savage, 1972) which, for example, implies the computational equivalence of feedforward and *convergent* recurrent networks up to a factor of $t^\star$ in size (Goldschlager & Parberry, 1986).

2.3.3 Finite Symmetric Recurrent Networks

Convergence Results: 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 whose value decreases along any productive computation path. 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 binary-state symmetric nets, sequential computations starting from any initial state terminate provided that $w(j,j) \geq 0$ for every $j \in V$ (such networks are sometimes called *semisimple*) (Hopfield, 1982). This result can be proved by using e.g. the Liapunov function

$$E(\mathbf{y}) = -\sum_{j=1}^{s} y_j \left(w_{j0} + \frac{1}{2} \sum_{i=1; i \neq j}^{s} w_{ji} y_i + w_{jj} y_j \right) . \tag{2.25}$$

Parallel computations of binary Hopfield nets can be shown to either reach a stable state, e.g. when the quadratic form introduced in equation 2.25 is negative definite (Goles, 1987; Goles-Chacc, Fogelman-Soulié, & Pellegrin, 1985), or eventually alternate between two different states (Bruck & Goodman, 1988; Goles & Olivos 1981a; Goles, 1987; Poljak & Sůra, 1983; Tchuente, 1986). These convergence results were further generalized for analog symmetric networks with activation function σ, which can be proved under mild hypotheses to converge to a fixed point or to a limit cycle of length at most two for parallel updates (Fogelman-Soulié, Mejia, Goles, & Martínez, 1989; Koiran, 1994) by applying a Liapunov function of the form 2.25 extended with an additive term (cf. equation 2.36)

$$\sum_{j=1}^{s} \int_{0}^{y_j} \sigma^{-1}(y) dy \,. \tag{2.26}$$

Convergence Time: The convergence time in Hopfield nets, i.e. the number of discrete-time updates before the network converges (with a given precision for analog states), has 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 (Haken & Luby, 1988) and parallel (Goles & Olivos, 1981b) modes. This is witnessed e.g. by symmetric, sequential or parallel implementations of a *binary counter* that traverses most of the network state space before it converges in time $\Omega(2^{s/8})$ asynchronous sequential updates (Haken, 1989) or $\Omega(2^{s/3})$ fully parallel steps (Goles & Martínez, 1989, 1990). On the other hand, a very fast average-case convergence of only $O(\log \log s)$ parallel update steps can be shown for binary Hopfield nets under reasonable assumptions (Komlós & Paturi, 1988).

However, the previous bounds do not take into account the size of the *weights*. An upper bound of $O(W)$ on the convergence time, where

$$W = \sum_{j,i \in V} |w_{ji}| \tag{2.27}$$

is the total *weight* of the network, follows from the characteristics of the energy function defined in equation 2.25 both for binary sequential (Fogelman-Soulié, Goles, & Weisbuch, 1983; Goles, 1985; Goles-Chacc, Fogelman-Soulié, & Pellegrin, 1985; Goles & Martínez, 1990) and parallel (Goles, 1987) Hopfield nets. For integer weights, this upper bound can be expressed more precisely as

$$\frac{\sum_{j=1}^{s} \sum_{i=1; i \neq j}^{s} |w_{ji}| + \sum_{j=1}^{s} |w_{j0} + e_j|}{2\left(1 + \min_{j \in V} w_{jj}\right)} \tag{2.28}$$

for sequential mode, or

$$\frac{1}{2}\left(\sum_{j=1}^{s} \sum_{i=1}^{s} |w_{ji}| + 3 \sum_{j=1}^{s} |w_{j0} + e_j| - s \right) \tag{2.29}$$

for parallel updates where

$$e_j = \begin{cases} 1 & \text{if } \sum_{i=0}^{s} w_{ji} \text{ is even} \\ 0 & \text{otherwise} \end{cases} \tag{2.30}$$

(Floréen, 1991). For more precise upper bounds with real weights see (Fogelman, Goles, & Weisbuch, 1983; Goles-Chacc, Fogelman-Soulié, & Pellegrin, 1985). These bounds yield polynomial-time convergence for binary symmetric networks with polynomial weights.

Moreover, the 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 of weights (Šíma, Orponen, & Antti-Poika, 2000). For binary symmetric networks which are described with M bits, convergence-time lower and upper bounds of $2^{\Omega(M^{1/3})}$ and $2^{O(M^{1/2})}$, respectively, have been shown. 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, a corresponding lower bound of $2^{\Omega(g(M))}$ updates has 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.

Stable States: As the Hopfield networks were originally proposed for use as associative memories, it is of interest to determine the number of their stable states, corresponding to the stored patterns. It has been shown that there are on the average asymptotically $1.05 \times 2^{0.2874s}$ many stable states in a binary Hopfield net of size s whose feedbacks and biases are zero ($w_{jj} = w_{j0} = 0$ for $j \in V$), and whose other weights are independent identically distributed zero-mean Gaussian random variables (McEliece, Posner, Rodemich, & Venkatesh, 1987; Tanaka & Edwards, 1980). 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) (Floréen & Orponen, 1989), two (Lipscomb, 1987), or three (Floréen & Orponen, 1989) stable states, is NP-complete. In fact, the problem of determining the exact number of stable states for a given binary Hopfield net is #P-complete (Floréen & Orponen, 1989; Lipscomb, 1987). (See appendix or (Garey & Johnson, 1979) for the definition of the complexity class #P.)

Also the problem of finding a stable state in a binary symmetric network is known to be complete (by so called logspace reductions) for the complexity class of polynomial-time local search problems PLS (Schäffer & Yannakakis, 1991); see appendix or (Johnson, Papadimitriou, & Yannakakis, 1988) for a precise definition of this class. Furthermore, the problem of computing the *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 back to it, is NP-hard (Floréen & Orponen, 1993). There is even no polynomial-time algorithm that approximates the attraction radius in a sequential or fully parallel binary Hopfield net to within a factor of $s^{1-\varepsilon}$ for any fixed $0 < \varepsilon \leq 1$, unless $P = NP$.

Minimum Energy Problem: Hopfield networks have also been applied to the fast approximate solution of combinatorial optimization problems (Hopfield & Tank, 1985). The cost function of an optimization problem is here encoded into the Liapunov function of a Hopfield network which is then minimized in the course of computation. Hence, the *MIN ENERGY problem* of finding a network state with minimal energy or energy less than a prescribed value for a given Hopfield net is of special interest. However, this problem is in general NP-complete for both binary (Barahona, 1982; Bertoni & Campadelli, 1994) and analog (Šíma, Orponen, & Antti-Poika, 2000) Hopfield nets.

On the other hand, for binary Hopfield nets whose architectures are planar lattices (Bieche, Maynard, Rammal, & Uhry, 1980) or planar graphs (Barahona, 1982) this issue is polynomial-time solvable. Furthermore, a polynomial-time *approximate* algorithm that solves the MIN ENERGY problem to within absolute error of less than $0.243W$ in binary Hopfield nets of weight W can be implemented (Šíma, Orponen, & Antti-Poika, 2000), based on a high-performance approximate algorithm for the MAX CUT problem (Goemans & Williamson, 1995; Mahajan & Ramesh, 1999). For $W = O(s^2)$, e.g. for symmetric networks with s binary neurons and constant weights, this result matches the lower bound of $\Omega(s^{2-\varepsilon})$ (Bertoni & Campadelli, 1994) which cannot be guaranteed by any approximate polynomial-time MIN ENERGY algorithm for every $\varepsilon > 0$, unless $P = NP$. In addition, the MIN ENERGY problem can be approximately solved to within absolute error $O(s/\log s)$ in polynomial time for special binary Hopfield nets whose architectures are two-level grids (Bertoni, Campadelli, Gangai, & Posenato, 1997).

Computational Power: Finally, we survey some results on the computational power of Hopfield nets (cf. section 2.3.2 for corresponding results for asymmetric networks). In the case of binary-state neurons it turns out that symmetric networks are actually capable of simulating arbitrary *convergent* asymmetric networks with only a linear overhead in time and size (Šíma, Orponen, & Antti-Poika, 2000). This linear-overhead simulation is an improvement of the original quadratic-size construction (Orponen, 1996). This tight converse to Hopfield's convergence theorem (Hopfield, 1982) thus shows in binary networks it holds in a quite strong sense that

$$\text{“convergence} \equiv \text{symmetry”},$$

that is not only do all binary symmetric network converge, but all convergent computations including those with asymmetric weights can be implemented efficiently in symmetric Hopfield nets. This had previously been known for acyclic threshold circuits that can be implemented by using only symmetric interconnections with the same architecture (Parberry, 1990).

However, in the case of finite binary neural acceptors for arbitrary-length input sequences, symmetric devices are properly weaker than finite automata (Hartley & Szu, 1987), and recognize a strict subclass of the regular languages called the *Hopfield languages* (Šíma & Wiedermann, 1998). More precisely, Šíma (1995) proved 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 able to faithfully recognize Hopfield languages (Šíma, 1997), which provides a lower bound on the computational power of finite analog Hopfield nets. Because of the above-mentioned convergence property (Koiran, 1994), finite analog Hopfield nets are unlikely to simulate arbitrary Turing machines. On the other hand, if fully parallel analog symmetric networks are augmented with an external oscillator that produces some infinite binary sequence containing 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 (Šíma, Orponen, & Antti-Poika, 2000). This implies that the results on the computational power of deterministic asymmetric networks summarized in Table 2 are still valid for Hopfield nets with an external oscillator of certain type. Especially for rational weights these devices are Turing universal. Thus this provides a full characterization of the computational power of finite analog-state discrete-time networks with rational weights in the form of

"Turing universality $\equiv$ asymmetric network $\equiv$ symmetric network + oscillator",

together with the necessary and sufficient condition that the external oscillator needs to satisfy in order to qualify for this equivalence.

2.3.4 Infinite Families of Binary Recurrent Networks

In section 2.3.1, we discussed finite neural acceptors, and their input protocol of entering the input sequentially bit by bit. Another approach to treating inputs of arbitrary lengths, analogous to the input convention used for feedforward networks in section 2.2, is to have an infinite family $\{N_n\}$ of binary-state recurrent networks, one N_n for each input length $n \geq 0$. Thus, for n-bit binary inputs $\mathbf{x} \in \{0,1\}^n$ a network N_n is used whose n input neurons are initialized accordingly in the initial network state. For recognizing a language $L \subseteq \{0,1\}^\star$, the respective network N_n is employed for an input word $\mathbf{x} \in \{0,1\}^n$ and, after it converges in time $t^\star$, its single output neuron *out* is read which indicates whether

weights	unbounded	polynomial
asymmetric	PSPACE/poly	PSPACE/poly
symmetric	PSPACE/poly	P/poly

Table 4: The computational power of polynomial-size families of binary recurrent networks.

$\mathbf{x}$ belongs to L, that is

$$y_{out}^{(t^\star)} = 1 \quad \text{iff} \quad \mathbf{x} \in L\,. \tag{2.31}$$

Within this context the *size* $S(n)$ can be defined as the number of units in N_n.

It is known that polynomial-size families of binary recurrent networks recognize exactly the languages in the complexity class PSPACE/poly (Lepley & Miller, 1983); see appendix or (Balcázar, Díaz, & Gabarró, 1995) for a definition. The above-mentioned equivalence between convergent asymmetric and symmetric networks gives the same result for polynomial-size families of Hopfield nets, i.e. they also recognize exactly PSPACE/poly (Orponen, 1996). In addition, if the Hopfield nets in these families are restricted to have *polynomial* symmetric weights (with respect to the input length), then their computational power reduces to P/poly. The dependence of computational power on the type of weights for polynomial-size families of binary networks is summarized in Table 4.

2.4 Probabilistic Networks

The computational properties of various stochastic versions of discrete-time perceptron networks have also been studied. A reference model of *probabilistic (stochastic) networks* can be defined by augmenting the respective deterministic model with additional random *binary* input units $i \in \Pi$, whose states in time represent 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 related by polynomial (in the model parameters) mutual simulations to other types of stochastic neural networks, such as those with unreliable computing states and connecting units (Siegelmann, 1999b; von Neumann, 1956); Boltzmann machines (Ackley, Hinton, & Sejnowski, 1985; Parberry, 1994; Parberry & Schnitger, 1989), etc.

Probabilistic networks can also be 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* by a network when its error probability is at most ε, $0 \leq \varepsilon < 1/2$, i.e., the probability that the 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 $\mathbf{x} \notin L$. Such symmetry in the probability of errors in accepting and rejecting an input can always be achieved by, e.g., adding a few random input units (Hajnal, Maass, Pudlák, Szegedy, & Turán, 1993).

We shall divide the analysis of probabilistic networks into section 2.4.1 for feedforward and section 2.4.2 for recurrent architectures.

2.4.1 Probabilistic Feedforward Networks

For simplicity consider first probabilistic binary-state feedforward networks (probabilistic threshold circuits). 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 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 reasonably large equivalent *deterministic* threshold circuit. Indeed, Parberry and Schnitger (1989) proved 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

$$\left\lceil \frac{8\varepsilon \ln 2}{(1-2\varepsilon)^2} + 1 \right\rceil ns + 1 \tag{2.32}$$

recognizing L.

Also, the complexity class TC^0_d ($d \geq 1$) associated with the families of threshold circuits has been generalized to its probabilistic version RTC^0_d which is the class of all languages $\varepsilon(n)$-recognized by families of polynomial-size and polynomial-weight probabilistic threshold circuits of depth d with an error given by a real sequence of probabilities

$$\varepsilon(n) = \frac{1}{2} - \frac{1}{n^{O(1)}}, \tag{2.33}$$

one for each input length $n \geq 0$. For example, the language IP that consists of all the input instances for which the Boolean inner product defined in equation 2.11 has value 1, has been shown to belong to RTC^0_2 (Hajnal, Maass, Pudlák, Szegedy, & Turán, 1993), thus confirming that probabilistic feedforward networks may be more efficient than their deterministic counterparts since $IP \notin TC^0_2$ (cf. section 2.2.1). On the other hand, at least for nonuniform circuit families at most one layer can be saved by introducing stochasticity in threshold circuits, since in this case it holds that

$$RTC^0_d \subseteq TC^0_{d+1} \tag{2.34}$$

for every $d \geq 1$ (Hajnal, Maass, Pudlák, Szegedy, & Turán, 1993) as depicted in Figure 2.

2.4.2 Probabilistic Recurrent Networks

Such a computational analysis has also been generalized to finite probabilistic recurrent networks with the saturated-linear activation function defined in equation 2.4 (Siegelmann, 1999b). The results are summarized and compared to the corresponding deterministic models in Table 2. Thus for integer weights, the results coincide with those for deterministic networks (see section 2.3.2), that is the binary-state probabilistic networks ε-recognize the regular languages.

Further, analog-state probabilistic networks with rational weights can ε-recognize in polynomial time exactly the languages from the nonuniform complexity class Pref-BPP/log, corresponding to polynomial-time bounded-error probabilistic Turing machines with so called prefix-closed logarithmic-length advice functions. (See the appendix or (Balcázar, Díaz, & Gabarró, 1995; Balcázar & Hermo, 1998) 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 to be rational numbers, the computational power of polynomial-time analog-state probabilistic networks with rational weights reduces to the recursive complexity class BPP.

Finally, $T(n)$-time bounded probabilistic recurrent networks with arbitrary real weights can be simulated in time $nT(n) + n^2$ by corresponding deterministic networks. Consequently, polynomial-time computations in this network model correspond to the complexity class P/poly.

2.5 Continuous-Time Dynamics

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

$$\frac{dy_j}{dt}(t) = -y_j(t) + \sigma(\xi_j(t)) = -y_j(t) + \sigma\left(\sum_{i=0}^{s} w_{ji} y_i(t)\right) \tag{2.35}$$

for $j = 1, \ldots, s$, where the excitation $\xi_j(t)$ of unit j is defined as in equation 2.1 and the saturated-linear activation function introduced in equation 2.4 is employed, implying that $\mathbf{y}(t) \in [0,1]^s$.

Using the Liapunov function

$$E(\mathbf{y}) = -\sum_{j=1}^{s} y_j \left(w_{j0} + \frac{1}{2}\sum_{i=1}^{s} w_{ji} y_i\right) + \sum_{j=1}^{s} \int_0^{y_j} \sigma^{-1}(y)dy \tag{2.36}$$

it can be shown that a continuous-time symmetric network (with $w_{ji} = w_{ij}$) conforming to this model converges from any initial state $\mathbf{y}(0)$ to some stable state satisfying $dy_j/dt = 0$ for all $j = 1, \ldots, s$ (Cohen & Grossberg, 1983; Hopfield 1984). Then the set of stable states of the corresponding discrete system described by equation 2.2 coincides with that of the continuous-time system introduced in equation 2.35. Also an exponential lower bound on the convergence time of such continuous-time Hopfield nets has been obtained in terms of the system dimension (Šíma & Orponen, 2001).

Moreover, for any given finite binary-state discrete-time recurrent network, a continuous-time asymmetric network controlled by dynamics equation 2.35 can be constructed that simulates the given discrete system faithfully, and has only a linear size overhead (Orponen, 1997c). In fact, such a simulation can even be achieved with continuous-time symmetric Hopfield nets (Šíma & Orponen, 2000, 2003). This implies that polynomial-size families of continuous-time (symmetric) networks are able to recognize at least the complexity class PSPACE/poly (see section 2.3.4).

3 Other Neural Network Models

Finally, we shall review the only recently analyzed computational capabilities of the RBF, winner-take-all, and spiking networks in sections 3.1, 3.2, and 3.3, respectively.

3.1 RBF Networks

The *RBF networks* represent an important alternative to the classical discrete-time perceptron networks. Units in RBF networks compute *radial basis functions*, which were introduced in the context of interpolation problems; see the survey by Powell (1985). In this case, the perceptron update rule 2.1 and 2.2 is modified so that the "excitation"

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

of unit j is proportional to the distance between the input vector $\mathbf{x}_j^{(t)} = (y_{ji_1}^{(t)}, \ldots, y_{ji_{n_j}}^{(t)}) \in \mathbf{R}^{n_j}$, consisting of the states of units $i_1, \ldots, i_{n_j} \in V$ incident on j, and unit j's *center* $\mathbf{w}_j = (w_{ji_1}, \ldots, w_{ji_{n_j}}) \in \mathbf{R}^{n_j}$, represented in terms of the corresponding "weights". In addition, a real positive "bias" parameter $w_{j0} > 0$ determines a *width* of RBF unit j. Usually the widths are the same for all units in the network, e.g. $w_{j0} = 1$ for $j = 1, \ldots, s$, although already two different width values in the RBF network provide provably more computational power than uniform widths (Schmitt, 2002). The new state of unit j at the next time instant

$t+1$ is determined similarly as in equation 2.2 but with an activation function $\sigma : \mathbf{R} \longrightarrow \mathbf{R}$ such as the *Gaussian function*

$$\sigma(\xi) = e^{-\xi^2}\,, \tag{3.2}$$

whose shape usually differs from that of sigmoidal functions (cf. equations 2.3–2.5).

For representing the binary values 0 and 1, two different analog states of RBF units are reserved. For this convention any Boolean NAND gate over multiple literals (input variables or their negations) can be implemented by an RBF unit employing the maximum norm ($\|\mathbf{x}\| = \|\mathbf{x}\|_\infty = \max_{i=1,\ldots,n} |x_i|$ for $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbf{R}^n$) in equation 3.1. For a large class of smooth activation functions σ, including the Gaussian function, which possess a special type of inflexion point ξ_0 (e.g. $\sigma''(\xi_0) = 0$ and $\sigma'(\xi_0)\sigma'''(\xi_0) < 0$ if σ''' exists), this NAND implementation is robust even when the analog representations of binary values are allowed to lie within specific disjoint small intervals around the exact real states (Šorel & Šíma, 2000).

It follows that many results described in section 2 concerning the computational power of perceptron networks can be reformulated also for RBF networks. For example, deterministic finite automata with q states can be implemented by recurrent networks with $O(\sqrt{q \log q})$ RBF units in a robust way (Šorel & Šíma, 2000). On the other hand, the Turing universality of finite RBF networks is 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 the competitive strategy used in many practical models, such as the Kohonen networks (Kohonen, 2001). The competition principle appears to be neurophysiologically plausible and has also efficient analog VLSI implementations (Yuille & Geiger, 2003). Thus, a k-WTA_n gate with n real inputs and n binary outputs computes a mapping k-$WTA_n : \mathbf{R}^n \longrightarrow \{0,1\}^n$, defined as

$$k - WTA_n(x_1, \ldots, x_n) = (y_1, \ldots, y_n) \tag{3.3}$$

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.

$$y_i = 1 \quad \text{iff} \quad |\{j;\, x_j > x_i,\, 1 \leq j \leq n\}| \leq k-1\,. \tag{3.4}$$

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 fewer than $\binom{n}{2} + n$ threshold gates (Maass, 2000), which on the other hand clearly suffice for WTA_n computation. This implies also that no *recurrent* perceptron network with $O(n)$ threshold gates can compute the WTA_n function in sublinear time $o(n)$.

Furthermore, any Boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$ in TC_2^0 (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 of n input variables (cf. equation 2.1) with polynomial natural (positive) weights (Maass, 2000).

Winner-take-all networks may also gain efficiency in the total wire length measure, as the function P_{LR}^k introduced in section 2.2.1 by equation 2.12 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 $O(k)$ (Legenstein & Maass, 2001). Hence, it appears that winner-take-all gates are more efficient than perceptrons.

3.3 Networks of Spiking Neurons

The most prominent position among neural network models proposed as alternatives 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 (Maass & Bishop, 1998). Their computational properties will first be surveyed for a deterministic model in section 3.3.1, and subsequently also a noisy version will be considered in section 3.3.2.

3.3.1 Deterministic Spiking Networks

The main feature differentiating spiking networks from perceptron-type models is their encoding of computational states as temporal differences between *spikes*, or *firing times* of neurons. Thus, suppose a sequence

$$0 \le y_j^{(1)} < y_j^{(2)} < \cdots < y_j^{(\tau)} < \ldots \tag{3.5}$$

of firing times is associated with each spiking neuron $j \in V$.

The spikes for the input neurons $V_{in} \subseteq V$ 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 $in \in V_{in}$, so that each bit is represented by the 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

$$y_{in}^{(2)} - y_{in}^{(1)} = \sum_{i=1}^{n} 2^{-i-c} x_i \tag{3.6}$$

(for sufficiently large integer constant $c > 0$) between the only two spikes $y_{in}^{(1)} < y_{in}^{(2)}$ of input neuron *in*.

In addition, for each connection from i to j a *response function* $\varepsilon_{ji} : \mathbf{R}_0^+ \longrightarrow \mathbf{R}$ ($\mathbf{R}_0^+$ denotes the set of nonnegative real numbers) is introduced which models the generic response $\varepsilon_{ji}(t)$ of neuron j to a spike from presynaptic unit i in continuous time $t \geq 0$ after the spike. For a discussion concerning the choice of the response functions see the following section that also includes examples of their simple shapes in Figure 3 (equations 3.18, 3.19). For notational simplicity, the response functions are formally defined to be zero in the case of missing connections, that is $\varepsilon_{ji}(t) \equiv 0$ when $w_{ji} = 0$. In addition, all weights w_{ji} associated with connections are assumed to be positive, i.e.

$$w_{ji} \geq 0 \quad \text{for all } \ j, i \in V\,. \tag{3.7}$$

Further, for each non-input unit $j \in V \setminus V_{in}$ a nonpositive *bias function* $w_{j0} : \mathbf{R}_0^+ \longrightarrow \mathbf{R}_0^- \cup \{-\infty\}$ determines its nonnegative threshold potential value

$$\Theta_j(t) = -w_{j0}(t) \geq 0 \tag{3.8}$$

at time $t \geq 0$ (passed since last firing) which, after being reached, causes neuron j to generate a spike as will be described in equation 3.12 below. Figure 4 shows an example of a simple bias function defined in equation 3.21. Note that the opposite function $\Theta_j = -w_{j0}$ called the *threshold function* is usually employed in this context. Nevertheless, in this article we use the bias function in order to differentiate it from the linear threshold function introduced in section 2.1.

Moreover, denote by

$$Y_j(t) = \left\{ y_j^{(\tau)} < t;\, \tau \geq 1 \right\} \tag{3.9}$$

the set of spikes of neuron j before a continuous time instant $t \geq 0$, and let

$$y_j(t) = \begin{cases} \max Y_j(t) & \text{for } \ Y_j(t) \neq \emptyset \\ t & \text{for } \ 0 \leq t \leq y_j^{(1)} \end{cases} \tag{3.10}$$

denote its last firing time for $Y_j(t) \neq \emptyset$; $y_j(t)$ is defined formally to be t before the first firing of j. Then an excitation

$$\xi_j(t) = w_{j0}(t - y_j(t)) + \sum_{i=1}^{s} \sum_{y \in Y_i(t)} w_{ji} \cdot \varepsilon_{ji}(t - y) \tag{3.11}$$

of neuron j at continuous time $t \geq y_j(t)$ (i.e. after its last firing $y_j(t)$) is defined as a weighted sum of j's responses to all preceding spikes from every neuron $i \in V$ incident on j (cf. equation 2.1). This includes the bias term $w_{j0}(t - y_j(t))$ which represents the opposite current threshold potential introduced in equation

3.8. Note that before j's first firing, $w_{j0}(t - y_j(t))$ corresponds to its initial value $w_{j0}(0) = -\Theta(0)$ according to equation 3.10. On the other hand, the excitation defined in equation 3.11 excluding the bias term $w_{j0}(t - y_j(t))$, is usually called the *potential* of neuron j.

The spikes $y_j^{(\tau)}$ $(\tau \geq 1)$ for each non-input neuron $j \in V \setminus V_{in}$ are determined recursively as follows. Formally choose $y_j^{(0)} < 0$ arbitrarily. Then the next firing time

$$y_j^{(\tau)} = \inf \left\{ t \geq 0;\, t > y_j^{(\tau-1)} \,\&\, \xi_j(t) \geq 0 \right\} \tag{3.12}$$

of neuron j for $\tau \geq 1$ is computed as the least time instant after j's last firing $y_j^{(\tau-1)}$ when its excitation reaches zero, that is when its potential reaches current threshold value defined in equation 3.8 (cf. equations 2.2, 2.3).

Finally, the spikes of output neurons encode the corresponding result of the computation. Here e.g. the same output protocol as for the input can be used.

Response Functions: 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 (Maass, 1996).

Each response function ε_{ji} associated to a connection from neuron i to neuron j is everywhere either nonnegative $\varepsilon_{ji} \geq 0$ or nonpositive $\varepsilon_{ji} \leq 0$, thus modeling biologically either an *excitatory* (EPSP) or an *inhibitory* (IPSP) *postsynaptic potential*. Furthermore,

$$\varepsilon_{ji}(t) = 0 \quad \text{for } \, t \in [0, \Delta_{ji}] \tag{3.13}$$

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 $\varepsilon^E, \varepsilon^I : \mathbf{R}_0^+ \longrightarrow \mathbf{R}$ with

$$\begin{aligned} \varepsilon^E(t) &= 0 \quad \text{for } \, t \geq t_{end}^E && (3.14) \\ \varepsilon^I(t) &= 0 \quad \text{for } \, t \geq t_{end}^I && (3.15) \end{aligned}$$

such that for every $j, i \in V$ and for all $t \geq 0$ the EPSP

$$\varepsilon_{ji}(\Delta_{ji} + t) = \varepsilon^E(t) \geq 0 \tag{3.16}$$

and the IPSP

$$\varepsilon_{ji}(\Delta_{ji} + t) = \varepsilon^I(t) \leq 0\,. \tag{3.17}$$

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 $0 < t_1 < t_2 < t_{end}^I$.

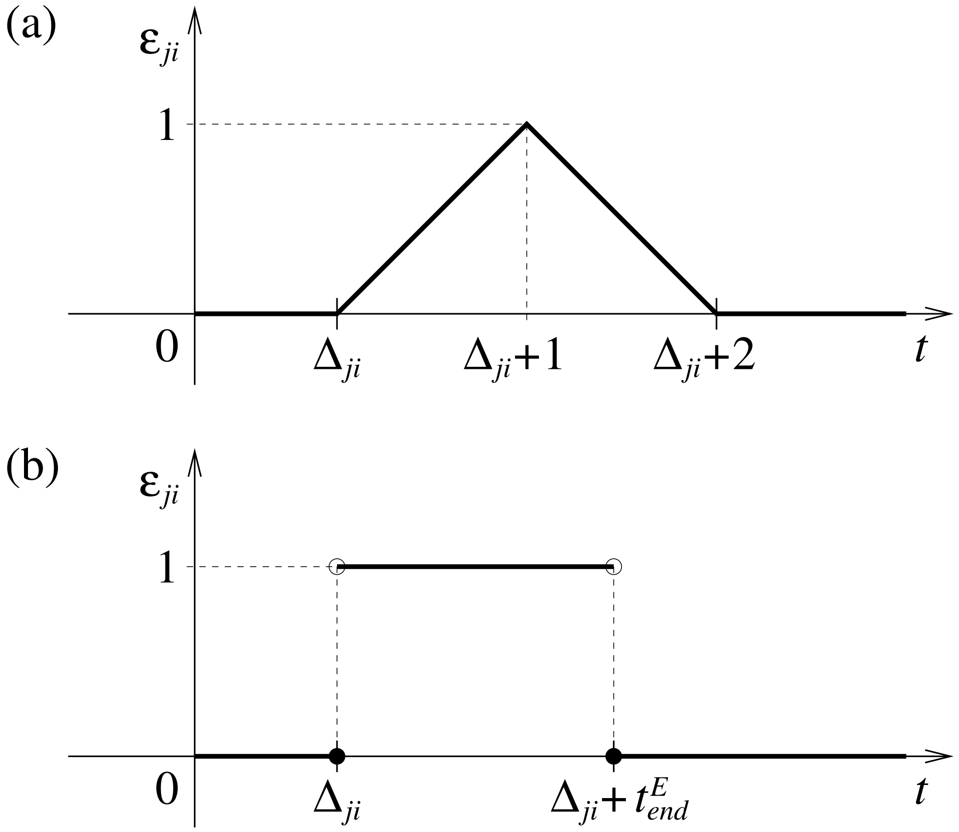

Figure 3: Examples of simple response functions: (a) piecewise linear and (b) piecewise constant functions.

Examples of possible EPSP functions include mathematically simple piecewise linear functions such as

$$\varepsilon_{ji}(t) = \begin{cases} 0 & \text{for } 0 \leq t \leq \Delta_{ji} \\ 1 - |t - \Delta_{ji} - 1| & \text{for } \Delta_{ji} < t < \Delta_{ji} + 2 \\ 0 & \text{for } t \geq \Delta_{ji} + 2 \end{cases} \tag{3.18}$$

depicted in Figure 3a. Sometimes not even the strict monotony of the EPSP functions is required (Maass & Ruf, 1999) and simple piecewise constant functions are employed, e.g.

$$\varepsilon_{ji}(t) = \begin{cases} 0 & \text{for } 0 \leq t \leq \Delta_{ji} \\ 1 & \text{for } \Delta_{ji} < t < \Delta_{ji} + t^E_{end} \\ 0 & \text{for } t \geq \Delta_{ji} + t^E_{end} \end{cases} \tag{3.19}$$

shown in Figure 3b. Examples of possible IPSP functions include, e.g. the functions $-\varepsilon_{ji}$, defined as the counterparts of the EPSP functions ε_{ji} by formula 3.18 or 3.19. It appears that the preceding general assumptions are also satisfied by more biologically plausible response functions.

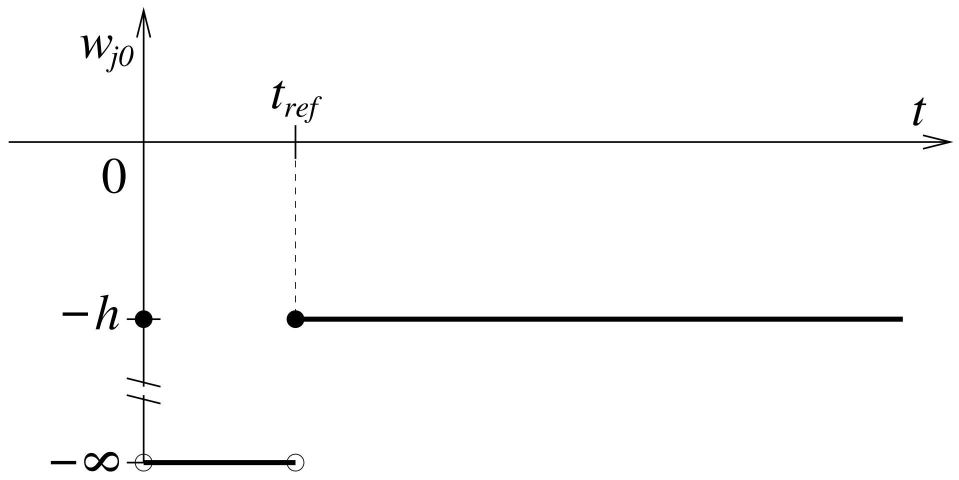

Figure 4: An example of simple bias function.

Bias Functions: A general bias function $w_0 : \mathbf{R}_0^+ \longrightarrow \mathbf{R}_0^- \cup \{-\infty\}$ is often employed so that

$$w_{j0}(t) = w_0(t) \quad \text{for every } \ j \in V \setminus V_{in} \,, \ \text{ and all } \ t \geq 0 \,. \tag{3.20}$$

It is assumed that $w_0(0) = -h$ for some constant $h > 0$, and $w_0(t) = -\infty$ for $t \in (0, t_{ref})$ in order to prevent neurons from firing in the *refractory period* t_{ref} following the previous spike. Further $w_0(t)$ returns to its initial value $-h$ after some time $t \geq t_{end}$. Examples of possible bias functions include mathematically simple piecewise constant functions such as

$$w_{j0}(t) = \begin{cases} -h & \text{for } \ t = 0 \\ -\infty & \text{for } \ 0 < t < t_{ref} \\ -h & \text{for } \ t \geq t_{ref} \end{cases} \tag{3.21}$$

depicted in Figure 4.

Single Spiking Neuron: We first consider a simplified model of only a single spiking neuron with n external inputs computing an n-variable Boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$. We assume that each input i ($1 \leq i \leq n$) is associated with weight w_i and the piecewise constant response function $\varepsilon_i(t)$ defined by formula 3.19 with delay parameter Δ_i and $t_{end}^E = 1$ corresponding to the EPSP, that is

$$\varepsilon_i(t) = \begin{cases} 0 & \text{for } \ 0 \leq t \leq \Delta_i \ \text{ or } \ t \geq \Delta_i + 1 \\ 1 & \text{for } \ \Delta_i < t < \Delta_i + 1 \,, \end{cases} \tag{3.22}$$

or its opposite $-\varepsilon_i(t)$ corresponding to the IPSP. Moreover, we assume here that the neuron has not fired for a while, so that its bias has returned to its initial value, i.e. the bias function $w_0(t) \equiv -h$ is constant. In the input protocol the

Computational Model	Simulating spiking network		
	Size	Weights	Simulation Time
Depth-d threshold circuit of size s	$O(s)$	unbounded	$O(d)$
	polynomial	$[0,1]$	
q-state deterministic finite automaton	$O(\sqrt{q})$	unbounded	constant period
	$O(\sqrt{q} \cdot \log q)$	$[0,1]$	
Turing machine with running time $T(n)$	finite	rational from $[0,1]$	$O(T(n))$

Table 5: Lower bounds on the computational power of spiking networks.

spike at the ith input at time $t = 0$ means $x_i = 1$ while $x_i = 0$ is encoded so that there is no firing at the ith input at all. Similarly, the spiking neuron outputs $f(x_1, \ldots, x_n) = 1$ iff it fires at any time as a result of computation on the input $x_1, \ldots, x_n$.

Maass and Schmitt (1999) observed that such a spiking neuron is strictly more powerful than a threshold gate (for $n \geq 3$) that can obviously be constructed as a special case by setting all the delays Δ_i equal. In addition, a spiking neuron can compute any Boolean function which can be written as a so called μ-DNF formula, i.e. a disjunctive normal form where each variable occurs at most once. On the other hand, a spiking neuron with n inputs can be implemented by a two-layered perceptron network having $n-1$ hidden threshold gates and an output OR gate (Schmitt, 1998). The number of hidden units in such a two-layered network, called the *threshold number* (Hammer, Ibaraki, & Peled, 1981), can here be lower bounded by $\lfloor n/2 \rfloor$. On the other, there exists a Boolean function with threshold number 2 (i.e. a disjunction of two threshold gates) that cannot be computed by a single spiking neuron (Schmitt, 1998).

The number of n-variable Boolean functions that can be computed by a spiking neuron is upper bounded by $2^{n^2+O(n \log n)}$, which together with the corresponding lower bound 2^{n^2} on the number of linear threshold functions mentioned in section 2.1 yields an estimate of approximately $2^{\Theta(n^2)}$ Boolean functions computable by a single unit (Maass & Schmitt, 1999).

Lower Bounds: We further review the lower bounds on the computational power of spiking networks due to Maass (1996) as they are summarized in Table 5. Thus, for a given feedforward circuit 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 the circuit's 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 a 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 network consisting of a finite number of spiking neurons with *rational weights* from $[0, 1]$ (while all the other parameters introduced in the assumptions on response and bias functions are also 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*.

Upper Bounds: On the other hand, the computational power of spiking networks can completely be characterized by introducing the upper bounds. Maass (1995) proved that the computational power of finite spiking networks with any piecewise linear response- and bias-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 defined in equations 2.3 and 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 (Maass, 1995) for further details. Thus, we obtain a full characterization of the computational power of finite spiking networks in the form of

"spiking network $\equiv$ discrete-time analog perceptron network $\equiv$ N-RAM",

provided that piecewise linear functions are involved. 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.

Piecewise Constant Response Functions However, if one restricts to the piecewise constant response functions (Maass & Ruf, 1999), e.g. such as defined in equation 3.19, which are easier to implement in hardware, then the spiking networks with piecewise monotone and continuous bias functions employing arbitrary real-valued parameters cannot carry out 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 bias functions with *rational* parameters are for online binary-string inputs computationally equivalent to deterministic finite automata for 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* with 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.

Different Synaptic Delays: 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 bias-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

$$CD_k(x_1,\ldots,x_k,x'_1,\ldots,x'_k) = 1 \quad \text{iff} \quad (\exists\, 1 \leq i \leq k)\;\; x_i = x'_i = 1 \tag{3.23}$$

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 (equation 2.3), or either $\Omega(\sqrt{k})$ or $\Omega(k^{1/4})$ units with piecewise polynomial (e.g. equation 2.4) or exponential (e.g. equation 2.5) activation functions, respectively (Maass, 1997c).

This result has further been generalized for *analog inputs* and arbitrary reasonable response functions. For example, the witnessing *element distinctness function* $ED_n : (\mathbf{R}_0^+)^n \longrightarrow \{0,1\}$ $(n \geq 2)$ which is defined as

$$ED_n(x_1,\ldots,x_n) = \begin{cases} 1 & \text{if } (\exists\, 1 \leq i < j \leq n)\;\; x_i = x_j \\ 0 & \text{if } (\forall\, 1 \leq i < j \leq n)\;\; |x_i - x_j| \geq 1 \\ \text{arbitrarily} & \text{otherwise}\,, \end{cases} \tag{3.24}$$

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 sigmoidal gates with the activation function defined in equation 2.5 (Maass, 1997c).

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 section 3.3.1 that assumes the precise firing times is not very realistic from that point of view. Thus, in the *noisy spiking networks* the excitation of spiking neuron j introduced in equation 3.11 just governs the probability that unit j fires. In particular, there are two functions $L, U : \mathbf{R} \times \mathbf{R}_0^+ \longrightarrow [0,1]$ 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 of spiking neuron j defined in equation 3.11 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, Maass (1996) has shown 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. Similar results have been achieved for analog computations (Maass, 1997b; Maass & Natschläger, 1997, 2000), e.g. the noisy spiking networks with temporal encoding of analog values can reliably simulate the feedforward perceptron networks with real inputs and outputs (Maass, 1997b).

4 Conclusion and Open Problems

We have presented a systematic survey of the known complexity theoretic properties of neural network models viewed from the perspective of digital computation. Several intriguing open questions remain. In this section we will conclude with a short evaluation of these models based on the results discussed in the paper. In particular, we will compare the models according to the taxonomy from Figure 1 and outline the main open problem areas.

4.1 Unit Type

We have considered several types of units. The main focus is on the traditional perceptron networks (section 2) since their complexity theoretic properties are best understood. A similar analysis for other unit types (section 3) is still not complete and their taxonomy should also be refined for different architectures, parameter domains, probabilistic computation, etc. as is clearly visible in Figure 1. In addition, several important open issues should be resolved, e.g. Turing universality of finite RBF networks, the power of recurrent WTA networks, etc. Nevertheless it appears that RBF networks are comparable to perceptron networks (section 3.1) while the WTA gates (section 3.2) may provably bring more efficient implementations of certain functions.

Also networks of spiking neurons are slightly, but not significantly more efficient than their perceptron counterparts (section 3.3). However, in addition to their increased biological plausibility, spiking neurons introduce temporal coding of computational states as a new source of efficient computation which undoubtedly deserves a deeper study.

4.2 Discrete vs. Continuous Time

The presented results have mainly been concerned with discrete-time dynamics. Thus apart from the spiking neurons that exploit the possibility of continuous time intervals for determining the firing times we only know that continuous-time perceptron networks are at least as powerful as the discrete-time models (section 2.5). In addition, the technique introduced within this context is somewhat unsatisfying, since the continuous-time computation is still basically discretized. We seem to be limited in our thinking by the discrete time mind-set of traditional complexity theory, which provides no adequate theoretical tools (e.g. complexity measures, reductions, universal computation) for "naturally" continuous-time computations (Moore, 1998; Orponen 1997a). Perhaps continuous-time neural networks will provide useful reference models for developing such tools. First steps along this direction have recently been established (Ben-Hur, Siegelmann, & Fishman, 2002; Gori & Meer, 2002).

4.3 Deterministic vs. Probabilistic Computation

Stochasticity represents an additional source of efficient computation in probabilistic (perceptron) networks (section 2.4.1). As we know for feedforward architectures there are functions that provably require three layers in efficient deterministic networks but can be implemented with only two layers in probabilistic networks (cf. Figure 2). Furthermore, Table 2 summarizing the results concerning the computational power of recurrent neural networks shows that the only difference between deterministic and probabilistic models consists in polynomial time computations with rational weights, which are characterized by the corresponding Turing complexity classes P and BPP. This means that from the computational power point of view stochasticity plays a similar role in neural networks as in conventional Turing computations. An interesting open question then concerns whether a more efficient implementation of finite automata by binary-state probabilistic neural networks can be achieved than that by deterministic threshold automata (Table 3).

4.4 Feedforward vs. Recurrent Architectures

Feedforward architectures (section 2.2) correspond to bounded computations and thus can only compete in computational power with convergent recurrent networks, or equivalently with symmetric networks. Nevertheless, we know that common interesting function (e.g. arithmetic operations) can be implemented efficiently with only a small number of layers, supporting the widely spread use of two- or three-layered networks in practical applications. A very important fact in this context is that two layers of perceptrons are not sufficient for an efficient implementation of certain functions. This is related to perhaps the most fascinating open problems concerning the TC hierarchies. Is the bounded-depth TC^0 hierarchy infinite, as is the corresponding AC^0 hierarchy (Håstad, 1989; Yao, 1985)? At the moment even the separation of TC_3^0 from TC_4^0 is open. Is $AC^0 \subseteq TC_d^0$ for some constant d? Also it is presently not known whether the inclusion $TC^0 \subseteq P$, or even $TC^0 \subseteq NP$ is proper.

On the other hand, the computational power of finite recurrent networks is nicely characterized by the descriptive complexity of their parameters (Tables 2, 5) and e.g. for rational weights these networks are Turing universal. However, more realistic models with fixed precision of real parameters or analog noise recognize only regular languages. Thus we can conclude that practical recurrent networks essentially represent efficient implementations of finite automata (e.g. Table 3).

4.5 Binary vs. Analog States

Analog-state neural networks prove to be at least as powerful and efficient computational devices as their binary-state counterparts. As concerns the feedforward networks the computational power of binary and analog states is almost equal, although we know that the number of neurons in a binary feedforward network can sometimes be reduced by a logarithmic factor when threshold gates are replaced by sigmoidal ones (section 2.2.2). In order to relate binary and analog feedforward networks more closely to each other, it would be of significance to extend the result of Maass, Schnitger, and Sontag (1991) on the equivalence of sigmoidal and threshold gate functions to cover also large-weight networks, i.e., to prove that for e.g. the activation functions σ given by formula 2.4 or 2.5, not just $TC_d^0(\sigma) = TC_d^0$ but even $LT_d(\sigma) = LT_d$ holds for all $d \geq 1$. Similar questions can be asked regarding, e.g. binding the spiking neuron hierarchies more tightly to the classical Boolean circuit hierarchies.

For recurrent architectures, one can theoretically exploit the analog states in finite networks to store infinite amounts of information, which drastically increases the computational power of analog networks from that of threshold automata to Turing universality or even more. However, as discussed above, this model, although theoretically elegant, is not very realistic from a practical point of view. Nevertheless, the result comparing the convergence times of binary and analog Hopfield nets in terms of representation size (section 2.3.3) suggests that analog models of computation may be worth investigating more for their efficiency gains than for their (theoretical) capability for arbitrary-precision real number computation. Again we can ask how efficient implementations of finite automata by analog neural networks can be achieved, and how this efficiency depends on the chosen activation function (cf. Table 3).

4.6 Symmetric vs. Asymmetric Weights

At least for binary-state perceptron networks we know that symmetric weights correspond to convergent dynamics in quite a strong sense, since not only do all Hopfield nets converge, but all convergent computations can be efficiently implemented using symmetric weights (section 2.3.3). An important special case of convergent asymmetric networks are those with feedforward architectures that can straightforwardly be implemented with symmetric weights. For analog states such an equivalence has not yet been established. In the case of recurrent analog networks, the power of asymmetric weights is characterized also by the condition that an external oscillator needs to satisfy in order to boost the power of symmetric networks to that of asymmetric ones (section 2.3.3). Without such an oscillator, symmetric networks cannot perform arbitrary unbounded computations, and thus are probably less powerful than finite automata.

The previous convergence results are valid only for perceptron networks. Another interesting problem area is to find conditions under which networks based on other types of units converge.

As has been presented in this survey, our understanding of the general computational power of neural network models has increased dramatically over the past 10–15 years, and appears now rather satisfactory. One of the main issue for further research is now to develop analysis and synthesis techniques for more *special-purpose* neural networks, in view of their applications in e.g. associative memory, trajectory following, or time series analysis.

Appendix

In order to make the survey more self-contained and comprehensible for readers of different backgrounds, we shall recall below the basic definitions of the complexity classes and related notions which have appeared in this paper. Nevertheless, we assume that the reader is at least familiar with the basic abstract computational models such as finite automata, Turing machines, etc. For further information see e.g. the textbooks by Balcázar, Díaz, and Gabarró (1995), Bovet and Crescenzi (1994), or Papadimitriou (1994).

$\mathbf{g} = \mathbf{O}(\mathbf{f}(\mathbf{n}))$: for some real $\varepsilon > 0$ and for all but finitely many natural n, $g(n) < \varepsilon \cdot f(n)$.

$\mathbf{g} = \mathbf{o}(\mathbf{f}(\mathbf{n}))$: for every real $\varepsilon > 0$ and for all but finitely many natural n, $g(n) < \varepsilon \cdot f(n)$.

$\mathbf{g} = \mathbf{\Omega}(\mathbf{f}(\mathbf{n}))$: for some real $\varepsilon > 0$ and for all but finitely many natural n, $g(n) > \varepsilon \cdot f(n)$.

$\mathbf{g} = \mathbf{\Theta}(\mathbf{f}(\mathbf{n}))$: $g = O(f(n))$ and $g = \Omega(f(n))$.

Recursive function: A function computable by an algorithm (Turing machine). A problem not solvable by any Turing machine is considered to be algorithmically undecidable.

P: The class of languages accepted by (deterministic) Turing machines in polynomial time, i.e. within a number of computation steps $T(n) = O(n^c)$ for some fixed c, where n is the length of input (e.g. the number of bits in its representation in a binary coding). Class P contains the problems which are believed to be computationally feasible.

NP: The class of languages accepted by nondeterministic Turing machines in polynomial time. At each computational step a nondeterministic Turing machine may choose its next move from a set of several (w.l.o.g. at most two) possibilities. It follows that on a given input there is not only one computation (path), but a set (a tree) of possible computations. According to the definition an input is accepted by a nondeterministic Turing machine iff there is at least one accepting computation in this set. In other words, class NP contains those problems whose solution can first be guessed (corresponds to a nondeterministic guess of an accepting computation from the set of possible computations) and then checked for correctness (whether this computation is accepting indeed) in polynomial time. For example, the class of satisfiable Boolean formulas SAT is in NP since for a given Boolean formula one can guess an assignment for each occurring variable and check in polynomial time whether this assignment satisfies the formula.

A is NP-hard: Any problem from NP can be reduced to A in polynomial time, i.e. for any B in NP there exists a function $\mathbf{f}$ computable in polynomial time that maps an input $\mathbf{x}$ for problem B to some input $\mathbf{f}(\mathbf{x})$ for problem A so that $\mathbf{x} \in B$ iff $\mathbf{f}(\mathbf{x}) \in A$. Hence, by solving only one NP-hard problem in polynomial time one would obtain polynomial time solutions for all problems in NP implying P=NP.

A is NP-complete: A is NP-hard and A in NP. Thus, the class of NP-complete problems contains the hardest problems in NP that are believed to be not computationally feasible in general. For example, the set of satisfiable Boolean formulas (in conjunctive normal form) SAT is NP-complete.

co-NP: The class of languages whose complements are accepted by polynomial-time nondeterministic Turing machines.

A is co-NP-complete: A is in co-NP and any problem from co-NP can be reduced to A in polynomial time.

PSPACE: The class of languages accepted by (deterministic) Turing machines working in polynomial space, i.e. by using a number of tape cells $S(n) = O(n^c)$ for some fixed c where n is the length of input. It holds that

$$P \subseteq NP \subseteq PSPACE\,, \qquad P \subseteq co-NP \subseteq PSPACE\,.$$

None of these inclusions are known to be proper at the moment, but all are conjectured to be so.

A is PSPACE-complete: A is in PSPACE and any problem from PSPACE can be reduced to A in polynomial time. The set of true quantified Boolean formulas (without free variables) QBF represents an example of PSPACE-complete problem.

BPP: The class of languages accepted by polynomial-time probabilistic Turing machines with an error probability bounded above by some positive constant $\varepsilon < 1/2$. A probabilistic Turing machine is a nondeterministic Turing machine that has exactly two different possible actions allowed at each computation step. In addition, on a given input the number of steps in all possible computations is equal while each such computation ends in a final state either accepting or rejecting the input. For any input $\mathbf{x}$ the error probability is defined by the ratio of computations on $\mathbf{x}$ giving wrong answer, to the total number of computations on $\mathbf{x}$. Class BPP is admitted to consist of computationally feasible problems and it holds that

$$P \subseteq BPP \subseteq PSPACE.$$

None of these inclusions are known to be proper.

#P: The class of integer valued functions f each corresponding to a nondeterministic Turing machine that has exactly $f(\mathbf{x})$ accepting computations on any input $\mathbf{x}$.

f is #P-complete: f is in #P and every function from #P can be computed in polynomial time when any value of f is granted within one computation step.

P/poly: The class of languages accepted by polynomial-time Turing machines with polynomial-length advice functions. An advice function $\mathbf{f}$, which does not need to be recursive, provides a Turing machine with an external advice, i.e. a string $\mathbf{f}(n)$ that depends only on the input length n. For a polynomial-length advice function $|\mathbf{f}(n)| = O(n^c)$ for some fixed c.

PSPACE/poly: The class of languages accepted by polynomial-space Turing machines with polynomial-length advice functions.

Pref-P/log: The class of languages accepted by polynomial-time Turing machines with logarithmic-length advice functions $\mathbf{f}$ ($|\mathbf{f}(n)| = O(\log n)$) that are closed under prefixes, i.e. $\mathbf{f}(n_1)$ is a prefix of $\mathbf{f}(n_2)$ for every $n_1 < n_2$.

Pref-BPP/log: The class of languages accepted by polynomial-time bounded-error probabilistic Turing machines with logarithmic-length advice functions closed under prefixes.

PLS: The class of polynomial-time local search problems. In a local search problem we are given a set of feasible solutions, each associated with an integer cost to be optimized and with a set of neighboring feasible solutions, and we want to find a locally optimal solution. A polynomial-time local search problem then assumes that in polynomial time one can produce an initial feasible solution, compute the cost of any feasible solution, and decide whether a feasible solution is locally optimal or find a better neighboring solution if it is not the case.

L is a definite language: There exist two finite sets of strings L_1 and L_2 such that $\mathbf{w} \in L$ iff either $\mathbf{w} \in L_1$ or $\mathbf{w} = \mathbf{v}_1\mathbf{v}_2$ for some prefix $\mathbf{v}_1$ and $\mathbf{v}_2 \in L_2$.

References

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

Allender, E. (1989). 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* (pp. 580–584). New York: IEEE Computer Society Press.

Alon, N. (1985). Asynchronous threshold networks. *Graphs and Combinatorics, 1*, 305–310.

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

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

Balcázar, J. L., Díaz, J., & Gabarró, J. (1995). *Structural Complexity I* (2nd ed.). Berlin: Springer-Verlag.

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

Balcázar, J. L., & Hermo, M. (1998). The structure of logarithmic advice complexity classes. *Theoretical Computer Science, 207*(1), 217–244.

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

Ben-Hur, A., Siegelmann, H. T., & Fishman, S. (2002). A theory of complexity for continuous time systems. *Journal of Complexity, 18*(1), 51–86.

Bertoni, A., & Campadelli, P. (1994). On the approximability of the energy function. In *Proceedings of the 4th International Conference on Artificial Neural Networks (ICANN'94)* (pp. 1157–1160). Berlin: Springer-Verlag.

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

Bieche, I., Maynard, R., Rammal, R., & Uhry, J. P. (1980). 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), 2553–2576.

Bovet, D. P., & Crescenzi, P. (1994). *Introduction to the Theory of Complexity.* Hemel Hempstead: Prentice Hall International (UK).

Bruck, J., & Goodman, J. W. (1988). A generalized convergence theorem for neural networks. *IEEE Transactions on Information Theory, 34*(5), 1089–1092.

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

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

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

Cover, T. M. (1965). Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers, 14*(3), 326–334.

Cover, T. M. (1968). Capacity problems for linear machines. In L. Kanal (Ed.), *Pattern Recognition* (pp. 283–289). Washington, DC: Thompson Book Co.

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

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

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

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

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

Floréen, P., & Orponen, P. (1994). Complexity Issues in Discrete Hopfield Networks. Research Report A–1994–4, Department of Computer Science, University of Helsinki.

Fogelman, F., Goles, E., & Weisbuch, G. (1983). Transient length in sequential iterations of threshold functions. *Discrete Applied Mathematics, 6*(1), 95–98.

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

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

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

Godbeer, G. H., Lipscomb, J., & Luby, M. (1988). 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.

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

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

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

Goldschlager, L. M., & Parberry, I. (1986). On the construction of parallel computers from various bases of Boolean functions. *Theoretical Computer Science, 43*(1), 43–48.

Goles, E. (1985). Dynamics of positive automata networks. *Theoretical Computer Science, 41*(1), 19–32.

Goles, E. (1987). Lyapunov functions associated to automata networks. In F. Fogelman-Soulié, Y. Robert, & M. Tchuente (Eds.), *Automata Networks in Computer Science—Theory and Applications* (pp. 58–81). Manchester University Press.

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

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

Goles, E., & Martínez, S. (1990). *Neural and Automata Networks: Dynamical Behavior and Applications.* Dordrecht: Kluwer Academic Publishers.

Goles, E., & Olivos, J. (1981a). Comportement periodique des fonctions a seuil binaires et applications. *Discrete Applied Mathematics, 3*(2), 93–105.

Goles, E., & Olivos, J. (1981b). The convergence of symmetric threshold automata. *Information and Control, 51*(2), 98–104.

Gori, M., & Meer, K. (2002). A step towards a complexity theory for analog systems. *Mathematical Logic Quarterly, 48*(1), 45–58.

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

Haken, A. (1989). Connectionist Networks That Need Exponential Time to Stabilize. Unpublished manuscript, Department of Computer Science, University of Toronto.

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

Hammer, P. L., Ibaraki, T., & Peled, U. N. (1981). Threshold numbers and threshold completions. In P. Hansen (Ed.), *Studies on Graphs and Discrete Programming, Annals of Discrete Mathematics, 11, Mathematics Studies, volume 59* (pp. 125–145). Amsterdam, North-Holland.

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

Håstad, J. (1989). Almost optimal lower bounds for small depth circuits. In S. Micali (Ed.), *Advances in Computing Research, Randomness and Computation, volume 5* (pp. 143–170). Greenwich, CT: JAI Press Inc.

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

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

Hofmeister, T. (1994). Depth-efficient threshold circuits for arithmetic functions. In V. P. Roychowdhury, K.-Y. Siu, & A. Orlitsky (Eds.), *Theoretical Advances in Neural Computation and Learning* (pp. 37–84). Boston: Kluwer Academic Publishers.

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

Hofmeister T., & Pudlák, P. (1992). A Proof That Division is Not in TC_2^0. Research Report No. 447, Dortmund University.

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

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

Hopfield, J. J., & Tank, D. W. (1985). "Neural" computation of decision in optimization problems. *Biological Cybernetics, 52*(3), 141–152.

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

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

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

Irmatov, A. A. (1996). Bounds for the number of threshold functions. *Discrete Mathematics and Applications, 6*(6), 569–583.

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

Kahn, J., Komlós, J., & Szemerédi, E. (1995). On the probability that a random $\{\pm 1\}^n$-matrix is singular. *Journal of the American Mathematical Society 8*(1), 223–240.

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

Kleene, S. C. (1956). Representation of events in nerve nets and finite automata. In C. E. Shannon, & J. McCarthy (Eds.), *Automata Studies, Annals of Mathematics Studies, volume 34*, (pp. 3–41). Princeton, NJ: Princeton University Press.

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

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

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

Komlós, J., & Paturi, R. (1988). Convergence results in an associative memory model. *Neural Networks, 1*(3), 239–250.

Legenstein, R. A, & Maass, W. (2001). Foundations for a circuit complexity theory of sensory processing. In T. K. Leen, T. G. Dietterich, & V. Tresp (Eds.), *Advances in Neural Information Processing Systems (NIPS 2000), volume 13* (pp. 259–265). Cambridge, MA: The MIT Press.

Lepley, M., & Miller, G. (1983). Computational Power for Networks of Threshold Devices in Asynchronous Environment. Technical Report, Department of Mathematics, MIT.

Lipscomb, J. (1987). On the Computational Complexity of Finding a Connectionist Model's Stable State Vectors. M.Sc. thesis, Department of Computer Science, University of Toronto.

Lupanov, O. B. (1961). Implementing the algebra of logic functions in terms of bounded depth formulas in the basis +,*,-. *Soviet Physics Doklady, 6*(2), 107–108.

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

Maass, W. (1995). On the computational complexity of networks of spiking neurons. In G. Tesauro, D. S. Touretzky, & T. K. Leen (Eds.), *Advances in Neural Information Processing Systems (NIPS'94), volume 7* (pp. 183–190). Cambridge, MA: The MIT Press.

Maass, W. (1996). On the computational power of noisy spiking neurons. In D. Touretzky, M. C. Mozer, & M. E. Hasselmo (Eds.), *Advances in Neural Information Processing Systems (NIPS'95), volume 8* (pp. 211–217). Cambridge, MA: The MIT Press.

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

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

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

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

Maass, W. (2000). On the computational power of Winner–Take–All. *Neural Computation, 12*(11), 2519–2536.

Maass, W., & Bishop, C. M. (Eds.) (1998). *Pulsed Neural Networks.* Cambridge, MA: The MIT Press.

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

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

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

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

Maass, W., & Schmitt, M. (1999). On the complexity of learning for spiking neurons with temporal coding. *Information and Computation, 153*(1), 26–46.

Maass, W., Schnitger, G., & Sontag, E. D. (1991). 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* (pp. 767–776). New York: IEEE Press.

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

McEliece, R. J., Posner, E. C., Rodemich, E. R., & Venkatesh, S. S. (1987). The capacity of the Hopfield associative memory. *IEEE Transactions on Information Theory, 33*(4), 461–482.

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

Minsky, M. L. (1967). *Computation: Finite and Infinite Machines.* Englewood Cliffs, NJ: Prentice Hall.

Minsky, M. L., & Papert, S. A. (1969). *Perceptrons.* Cambridge, MA: The MIT Press.

Moore, C. (1998). Finite-dimensional analog computers: flows, maps, and recurrent neural networks. In *Proceedings of the 1st International Conference on Unconventional Models of Computation* (pp. 59–71). Berlin: Springer-Verlag.

Muroga, S. (1971). *Threshold Logic and its Applications.* New York: Wiley-Interscience.

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

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

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

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

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

Orponen, P. (1997a). A survey of continuous-time computation theory. In D.-Z. Du, & K.-I. Ko (Eds.), *Advances in Algorithms, Languages, and Complexity* (pp. 209–224). Dordrecht: Kluwer Academic Publishers.

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

Orponen, P. (1997c). 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* (pp. 86–103). Berlin: Springer-Verlag.

Orponen, P. (2000). 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, "AI of Tomorrow": Symposium on Theory, volume 3* (pp. 89–96). Vaasa, Finland: Finnish AI Society.

Papadimitriou, C. H. (1994). *Computational Complexity.* Reading, MA: Addison-Wesley.

Parberry, I. (1990). A primer on the complexity theory of neural networks. In R. B. Banerji (Ed.), *Formal Techniques in Artificial Intelligence: A Sourcebook, Studies in Computer Science and Artificial Intelligence, volume 6* (pp. 217–268). Amsterdam: Elsevier – North-Holland.

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

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

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

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

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

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

Razborov, A. A. (1992). On small depth threshold circuits. In O. Nurmi, & E. Ukkonen (Eds.), *Proceedings of the 3rd Scandinavian Workshop on Algorithm Theory (SWAT'92), Helsinki, Finland, LNCS 621* (pp. 42–52). Berlin: Springer-Verlag.

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

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

Roychowdhury, V. P., Siu, K.-Y., & Orlitsky, A. (Eds.) (1994). *Theoretical Advances in Neural Computation and Learning.* Boston: Kluwer Academic Publishers.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature, 323*, 533–536.

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

Savage, J. E. (1998). *Models of Computation: Exploring the Power of Computing.* Reading, MA: Addison-Wesley.

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

Schläfli, L. (1901). *Theorie der vielfachen Kontinuität.* Zürich: Zürcher & Furrer.

Schmitt, M. (1998). On computing Boolean functions by a spiking neuron. *Annals of Mathematics and Artificial Intelligence, 24*(1-4), 181–191.

Schmitt, M. (2002). Descartes' rule of signs for radial basis function neural networks. *Neural Computation, 14*(12), 2997–3011.

Siegelmann, H. T. (1994). On the computational power of probabilistic and faulty neural networks. In S. Abiteboul, & E. Shamir (Eds.), *Proceedings of the 21st International Colloquium on Automata, Languages, and Programming (ICALP'94), LNCS 820* (pp. 23–34). Berlin: Springer-Verlag.

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

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

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

Siegelmann, H. T., Roitershtein, A., & Ben-Hur, A. (2000). Noisy neural networks and generalizations. In S. A. Solla, T. K. Leen, & K.-R. Müller (Eds.), *Advances in Neural Information Processing Systems (NIPS'99), volume 12* (pp. 335–341). Cambridge, MA: The MIT Press.

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

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

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

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

Šíma, J. (2001). The computational capabilities of neural networks (extended abstract). In *Proceedings of the 5th International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA'01), Prague, Czech Republic* (pp. 22–26). Vienna: Springer-Verlag.

Šíma, J., & Orponen, P. (2000). 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* (pp. 36–42). Wetaskiwin, Canada: ICSC Academic Press.

Šíma, J., & Orponen, P. (2001). Exponential transients in continuous-time symmetric Hopfield nets. In *Proceedings of the 11th International Conference on Artificial Neural Networks (ICANN'01), Vienna, Austria, LNCS 2130* (pp. 806–813). Berlin: Springer-Verlag.

Šíma, J., & Orponen, P. (2003). Continuous-time symmetric Hopfield nets are computationally universal. *Neural Computation, 15*(3), 693–733.

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

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

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

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

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

Siu, K.-Y., Roychowdhury, V. P., & Kailath, T. (1993). Computing with almost optimal size neural networks. In S. J. Hanson, J. D. Cowan, & C. L. Giles (Eds.), *Advances in Neural Information Processing Systems (NIPS'92), volume 5* (pp. 19–26). San Mateo: Morgan Kaufmann.

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

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

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

Šorel, M., & Šíma, J. (2000). 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* (pp. 431–439). Berlin: Springer-Verlag.

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

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

Vollmer, H. (1999). *Introduction to Circuit Complexity.* Berlin: Springer-Verlag.

von Neumann, J. (1956). Probabilistic logics and the synthesis of reliable organisms from unreliable components. In C. E. Shannon, & J. McCarthy (Eds.), *Automata Studies, Annals of Mathematics Studies, volume 34* (pp. 43–98). Princeton, NJ: Princeton University Press.

Wegener, I. (1987). *The Complexity of Boolean Functions.* Chichester: Wiley/Teubner. Available electronically at `http://www.eccc.uni-trier.de/eccc-local/ECCC-Books/wegener_book_readme.html`.

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

Werbos, P. J. (1974). Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. Ph.D. thesis, Harvard University.

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

Yao, A. C.-C. (1985). Separating the polynomial–time hierarchy by oracles. *Proceedings of the 26th Annual Symposium on the Foundations of Computer Science (FOCS'85), Portland, Oregon* (pp. 420–428). New York: IEEE Computer Society.

Yuille, A. L., & Geiger, D. *Winner-take-all networks.* In Arbib, M. A. (Ed.), *The Handbook of Brain Theory and Neural Networks, 2nd Ed* (pp. 1228–1231). Cambridge, MA: The MIT Press.