Lower Bounds for the Computational Power of Networks of Spiking Neurons

Wolfgang Maass

Institute for Theoretical Computer Science
Technische Universitaet Graz
Klosterwiesgasse 32/2
A-8010 Graz, Austria
e-mail: maass@igi.tu-graz.ac.at

June 19, 1995

Abstract

We investigate the computational power of a formal model for networks of spiking neurons. It is shown that simple operations on phase-differences between spike-trains provide a very powerful computational tool that can in principle be used to carry out highly complex computations on a small network of spiking neurons. We construct networks of spiking neurons that simulate arbitrary threshold circuits, Turing machines, and a certain type of random access machines with real valued inputs. We also show that relatively weak basic assumptions about the response- and threshold-functions of the spiking neurons are sufficient in order to employ them for such computations.

1 Introduction and Basic Definitions

There exists substantial evidence that timing phenomena such as temporal differences between spikes and frequencies of oscillating subsystems are integral parts of various information processing mechanisms in biological neural systems (for a survey and references see e.g. Kandel et al., 1991; Abeles, 1991; Churchland and Sejnowski, 1992; Aertsen, 1993). Furthermore simulations of a variety of specific mathematical models for networks of spiking neurons have shown that temporal coding offers interesting possibilities for solving classical benchmark-problems such as associative memory, binding, and pattern segmentation (for an overview see Gerstner et al., 1993). Very recently one has also started to build *artificial* neural nets that model networks of spiking neurons (see e.g. Murray and Tarassenko, 1994, Watts, 1994). Some aspects of these models have also been studied analytically (see e.g. Gerstner and van Hemmen, 1994, Gerstner, 1995), but almost nothing is known about their computational complexity (see Judd and Aihara, 1993, for some first results in this direction). In this article we investigate a simple formal model SNN for networks of spiking neurons that allows us to model the most important timing phenomena of neural nets, and we prove lower bounds for its computational power.

Quite a number of different mathematical models for networks of spiking neurons have previously been introduced within the frameworks of theoretical physics and theoretical neurobiology (see e.g. Lapicque, 1907, Buhmann and Schulten, 1986, Crair and Bialek, 1990, Gerstner, 1991, Gerstner, Ritz, van Hemmen, 1993; for a survey and the relationship between these and related models see e.g. Tuckwell, 1988, and Gerstner, 1995). The computational model SNN that we consider in this article is most closely related to the *spike response model* of Gerstner, 1991, and Gerstner, Ritz, van Hemmen, 1993. Similarly as in Buhmann and Schulten, 1986, we consider in this article only the deterministic case (which corresponds to the limit case $\beta \rightarrow \infty$ for the inverse temperature β in the spike response model, respectively the noise-free case). However in contrast to these preceding models we do not fix particular (necessarily somewhat arbitrarily chosen) response- and threshold-functions in our model SNN. Instead, we want to have the possibility to use the SNN-model as a framework for *investigating* the computational power of various *different* response- and threshold-functions. In addition, we would like to make sure that various *different* response- and threshold-functions that are observed in specific biological neural systems are in fact special cases of the response- and threshold-functions in the here considered formal model SNN.

Definition of a Spiking Neuron Network (SNN): *An SNN $\mathcal{N}$ consists of*

- *a finite directed graph $\langle V, E\rangle$ (we refer to the elements of V as "<u>neurons</u>" and to the elements of E as "<u>synapses</u>")*
- *a subset $V_{in} \subseteq V$ of <u>input neurons</u>*
- *a subset $V_{out} \subseteq V$ of <u>output neurons</u>*

– *for each neuron $v \in V - V_{in}$ a* <u>*threshold-function*</u> $\Theta_v : \boldsymbol{R}^+ \to \boldsymbol{R} \cup \{\infty\}$ *(where $\boldsymbol{R}^+ := \{x \in \boldsymbol{R} : x \geq 0\}$)*

– *for each synapse $\langle u, v\rangle \in E$ a* <u>*response-function*</u> $\varepsilon_{u,v} : \boldsymbol{R}^+ \to \boldsymbol{R}$ *and a* <u>*weight-function*</u> $w_{u,v} : \boldsymbol{R}^+ \to \boldsymbol{R}$.

We assume that the firing of the input neurons $v \in V_{in}$ is determined from outside of $\mathcal{N}$, i.e. the sets $F_v \subseteq \boldsymbol{R}^+$ of firing times ("spike trains") for the neurons $v \in V_{in}$ are given as the <u>*input*</u> *of $\mathcal{N}$. Furthermore we assume that a set $T \subseteq \boldsymbol{R}^+$ of* <u>*potential firing times*</u> *has been fixed (we will consider only the cases $T = \boldsymbol{R}^+$ and $T = \{i \cdot \mu \ : \ i \in \boldsymbol{N}\}$ for some $\mu > 0$).*

For a neuron $v \in V - V_{in}$ one defines its set F_v of <u>*firing times*</u> *recursively. The first element of F_v is $\inf\{t \in T : P_v(t) \geq \Theta_v(0)\}$, and for any $s \in F_v$ the next larger element of F_v is $\inf\{t \in T : t > s \text{ and } P_v(t) \geq \Theta_v(t-s)\}$, where the* <u>*potential function*</u> *$P_v : \boldsymbol{R}^+ \to \boldsymbol{R}$ is defined by*

$$P_v(t) := 0 + \sum_{u \,:\, \langle u,v\rangle \in E} \ \sum_{s \in F_u \,:\, s < t} w_{u,v}(s) \cdot \varepsilon_{u,v}(t-s)$$

(the trivial summand 0 makes sure that $P_v(t)$ is well-defined even if $F_u = \phi$ for all u with $\langle u, v\rangle \in E$).

The firing times ("spike trains") F_v of the output neurons $v \in V_{out}$ that result in this way are interpreted as the <u>*output*</u> *of $\mathcal{N}$.*

Regarding the set T of potential firing times we consider in this article primarily the case $T = \mathbf{R}^+$ (<u>SNN with continuous time</u>), and only in Corollary 2.5 the case $T = \{i \cdot \mu : i \in \mathbf{N}\}$ for some $\mu > 0$ (<u>SNN with discrete time</u>).

Our subsequent assumptions about the threshold functions Θ_v will imply that for each SNN $\mathcal{N}$ there exists a bound $\tau_{\mathcal{N}} \in \mathbf{R}$ with $\tau_{\mathcal{N}} > 0$ such that $\Theta_v(x) = \infty$ for all $x \in (0, \tau_{\mathcal{N}})$ and all $v \in V - V_{in}$ ($\tau_{\mathcal{N}}$ may be interpreted as the minimum of all "refractory periods" τ_{ref} of neurons in $\mathcal{N}$). Furthermore we assume that all "input spike trains" F_v with $v \in V_{in}$ satisfy $|F_v \cap [0,t]| < \infty$ for all $t \in \mathbf{R}^+$. On the basis of these assumptions one can also in the continuous case easily show that the firing times are well-defined for all $v \in V - V_{in}$ (and occur in distances of at least $\tau_{\mathcal{N}}$).

In models for *biological neural systems* one assumes that if x time-units have passed since its last firing, the current threshold $\Theta_v(x)$ of a neuron v is "infinite" for $x < \tau_{ref}$ (where τ_{ref} = refractory period of neuron v), and then approaches quite rapidly from above some constant value. A neuron v "fires" (i.e. it sends an "action potential" or "spike" along its axon) when its current membrane potential $P_v(t)$ at the axon hillock exceeds its current threshold Θ_v. $P_v(t)$ is the sum of various postsynaptic potentials $w_{u,v} \cdot \varepsilon_{u,v}(t-s)$. Each of these terms describes an *excitatory* (EPSP) or *inhibitory* (IPSP) *postsynaptic potential* at the axon hillock of neuron v at time t, as a result of a spike that had been generated by the "presynaptic" neuron u at time s, and which has been transmitted through a synapse between both neurons. Recordings of an EPSP typically show a function that has a constant value c (c = resting membrane potential; e.g. $c = -70mV$) for some initial time-interval (reflecting the axonal and synaptic transmission time), then rises to a peak-value, and finally drops back to the same constant value c. An IPSP tends to have the negative shape of an EPSP (see Figure 3 in section 2). For the sake of mathematical simplicity we assume in the SNN-model that the constant initial and final value of all response-functions $\varepsilon_{u,v}$ is equal to 0 (in other words: $\varepsilon_{u,v}$ models the *difference* between an action potential and the resting membrane potential c). Different presynaptic neurons u generate postsynaptic potentials of different sizes at the axon hillock of a neuron v, depending on the size, location and current state of the synapse (or synapses) between u and v. This effect is modelled by the weight-factors $w_{u,v}(s)$.

The precise shapes of threshold-, response-, and weight-functions may vary among different biological neural systems, and even within the same system. Fortunately one can prove significant *upper bounds* for the computational complexity of SNN's $\mathcal{N}$ without *any* assumptions about the *specific shapes* of these functions of $\mathcal{N}$. Instead, for such upper bounds one only has to assume that they are of a reasonably simple *mathematical structure* (see Maass, 1994b, 1995c).

In order to prove *lower bounds* for the computational complexity of a SNN $\mathcal{N}$ one is forced to make more specific assumptions about these functions. However we show in this article that significant (and in some cases *optimal*, see section 3) lower bounds can be shown under some rather weak *basic assumptions* about these functions, which will be further relaxed in section 4. These basic assumptions (see section 2) mainly require that EPSP's have an arbitrarily small time-segment where they increase linearly, and some arbitrarily small time-segment where they decrease linearly. Since the computational power of SNN's may potentially increase through the use of time-dependent weights, *lower* bounds for their computational power are more significant if they do *not* involve the use of time-dependent weights. Hence we will assume throughout this article that *all weight-functions $w_{u,v}(s)$ have a constant value $w_{u,v}$ which does not depend on the time s.*

Apart from the abovementioned condition on the existence of linear segments in EPSP's, the basic assumptions which underlie the lower bound results of this article involve no other significant conditions on the shape of response- and threshold-functions. Hence one may argue that these basic assumptions are biologically plausible. In addition we will show in section 4 that the same lower bounds can be shown if also phenomena such as "adaption" of neurons, or a "reset" of the potential after a firing are taken into account. Thus the more critical points with regard to the biological interpretation of these lower bound results appear to be the relatively simple firing mechanism of the SNN-model, which for example ignores for the sake of simplicity nonlinear interactions among postsynaptic potentials such as integration of potentials within the dendritic tree of a neuron, and various possible sources of "imprecision" in the determination of the firing times. The latter issue can partially be taken into account by considering the variation of the SNN-model with *discrete* firing times as in Corollary 2.5 (although the implicit global synchronization of this version is not completely satisfactory). In this variation of the SNN-model with discrete firing times $i \cdot \mu$ for $i \in \mathbf{N}$ one can view a firing of a neuron at time $i \cdot \mu$ as representing a somewhat imprecise firing time in a small interval *around* time $i \cdot \mu$.

The computational complexity of another neural network model where timing plays an important role has previously been investigated by Judd and Aihara, 1993. Their model PPN is also motivated by biological spiking neurons, but it employs a quite different firing mechanism. There are no response-functions in this model, and instead of integrating all incoming EPSP's and IPSP's in order to determine whether it should "fire", a neuron in a PPN randomly selects a *single* one of the incoming "stimulations" of maximal size, and determines on the basis of that stimulation whether it should fire. Consequently, computations in this model PPN proceed quite differently from computations in models of spiking neurons such as the spike response model of Gerstner and van Hemmen, 1994, or the here considered model SNN. Judd and Aihara, 1993, construct PPN's which can simulate Turing machines that use at most a *constant* number s of cells on their tapes, where s is bounded by the number of neurons in the simulating PPN. However a Turing machine with a constant bound s on its number of tape cells is just a special case of a finite automaton, and hence this result does not show that a PPN of finite size can have the computational power of an arbitrary Turing machine. In contrast to the quoted result about PPN's, it is shown in Theorem 2.1 of this article that with arbitrary response- and threshold-functions which satisfy the basic assumptions of section 2 one can construct for any given Turing machine M an SNN $\mathcal{N}_M$ of *finite size* that can simulate *any* computation of M in real-time (even if the number of tape cells that M uses is much larger than the number of neurons in $\mathcal{N}_M$). In addition, at the end of section 4 we will describe a way how a simulation of arbitrary Turing machines can also be accomplished by finite SNN's whose response- and threshold-functions are *piecewise constant*. If we understand the model of Judd and Aihara, 1993, correctly (their description is somewhat unclear), then our method for proving this can also be used to show that with the help of a module which decides whether two neurons have fired simultaneously, one can simulate (although not in real-time) *any* Turing machine M (where M may use an *unbounded* number of tape cells) by some PPN $\mathcal{P}_M$ of *finite* size, thereby improving the lower bound for the computational power of PPN's due to Judd and Aihara, 1993, from finite automata to Turing machines.

The focus in the investigation of computations in biological neural systems differs in two essential aspects from that of classical computational complexity theory. First, the timing constraints for computations in biological neural systems are often tighter than for computations in digital computers, and many complex computations have to be carried out in "real-time" with relatively slow "switching elements". Secondly, one is not only interested in separate computations on unrelated inputs, but also in the ability of the system to *learn* to react appropriately to a *sequence* of *related* tasks.

Hence the custom to evaluate the computational power in terms of "complexity classes" such as P or $P/poly$ appears to be less suitable for the investigation of models for biological neural systems, and we therefore resort to an analysis in terms of refined concepts such as "real-time computations" and "real-time simulations". In this way we get not only information about the relationship between the "large scale" complexity classes (e.g. polynomial time) for these models for biological neural systems, but also about their behaviour in terms of common notions of "low-level complexity" such as sublinear or real-time.

Furthermore with the help of our refined analysis of real-time simulations one also gets information about the "adaptive" or "learning" abilities of the considered models. Assume for example that $(\langle x(i), y(i)\rangle)_{i\in\mathbf{N}}$ is the protocol of some real-time "learning process" of a system M, where the $y(i)$ are the "responses" of M to a sequence $x(i)$ of "stimuli". If one has shown that another model M' can simulate M in real-time, then this entails that the same "learning process" can also be carried out in real-time by M'.

Definition of real-time computation and real-time simulation

Fix some arbitrary (finite or infinite) input alphabet A_{in} and output-alphabet A_{out} (for example they can be chosen to be $\{0,1\}$, $\{0,1\}^*$ or $\boldsymbol{R}$). We say that a machine M processes a sequence $(\langle x(i), y(i)\rangle)_{i\in\boldsymbol{N}}$ of pairs $\langle x(i), y(i)\rangle \in A_{in} \times A_{out}$ in real-time r, if M outputs $y(i)$ for every $i \in \boldsymbol{N}$ within r computation steps after having received input $x(i)$ (for $i > 0$ we assume that $x(i)$ is presented at the next step after M has given output $y(i-1)$).

We say that a machine M' simulates a machine M in real-time (with delay-factor Δ) if for every $r \in \boldsymbol{N}$ and every sequence that is processed by M in real-time r, M' can process the same sequence in real time $\Delta \cdot r$.

In the case of SNN's M we count each spike in M as a computation step.

We first would like to point out that these notions contain the usual notions of a computation respectively simulation as special cases. Let $\{0,1\}^*$ be the set of all binary sequences of finite length. If M computes a boolean function $F : \{0,1\}^* \to \{0,1\}$ in time $t(n)$ (in the usual sense of computational complexity theory), then one can identify each input $\langle z_1, \ldots, z_n\rangle \in \{0,1\}^*$ with an infinite sequence $(x(i))_{i\in\mathbf{N}}$ where $x(i) = z_i$ for $i \leq n$ and $x(i) = B$ for $i > n$ (assume that M gets one input bit per step, $B :=$ "blank"). Furthermore one can set $y(i) = B$ for those steps i where M's computation is not yet finished, and $y(i) = F(\langle z_1, \ldots, z_n\rangle)$ for all later i (in particular for all $i \geq t(n)$). Obviously M processes this sequence $(\langle x(i), y(i)\rangle)_{i\in\mathbf{N}}$ in real-time 1. Hence, if another machine M' can simulate M in real-time with delay-factor Δ, then M' can compute the same function $F : \{0,1\}^* \to \{0,1\}$ in time $\Delta \cdot t(n)$. This implies that a real-time simulation is a special case of a linear-time simulation. In particular, every computational problem that can be solved by M within a certain time complexity, can be solved by M' within the same time complexity (up to a constant factor).

In addition, the remarks before the definition imply that when we show that M' can simulate M in *real-time*, we may conclude that any *adaptive behavior* (or *learning* algorithm) of M can also be implemented on M'. Finally we would like to point out that for the investigation of specific computational- and learning-problems on specific models for biological neural nets one would like to get eventually also estimates for the *size* of the constant r in real-time processing, respectively the *size* of the delay-factor Δ in a real-time simulation. Such refined analysis (which will not be carried out in this paper) appears to be also of interest, since it is likely to throw some light on the specific advantages and disadvantages of different models for biological neural systems (e.g. networks of spiking neurons versus analog neural nets), which are shown in Maass 1994b, 1995c, to be equivalent with regard to the preceding notion of a real-time simulation.

In contrast to the usual notion of a simulation, a *real-time* simulation of another computational model M by an SNN implies that the simulation of each computation step of M requires only a *fixed* number of spikes in the SNN. In particular the required number of spikes does not become larger for the simulation of later computation steps of M.

Input- and Output-Conventions

For simulations between SNN's and Turing machines one may either assume that the SNN gets an input (or produces an output) from $\{0,1\}^*$ in the form of a spike-train (i.e. one bit per unit of time), or that the input (output) of the SNN is encoded into the phase-difference of just two spikes. The former convention is suitable for comparisons with Turing machines that receive at each computation step a single input bit and produce a single output bit. For comparisons with Turing machines that start with the whole input written on a specified tape, and have their whole output written on another tape when the machine halts, it is more adequate to assume that the SNN receives at the beginning of a computation the whole tape content of the input tape encoded into the time-difference φ between two spikes (using the same encoding as we will use in section 2 in order to represent the content of a stack), and that the SNN also provides the final content of the output tape in the same form. *Real-valued* input or output for an SNN is always encoded into the phase-difference of two spikes.

Notation

We employ in this article the following common notation: We write $\mathbf{N}$ for the set of natural numbers (including 0), $\mathbf{Q}$ for the set of rational numbers, and $\mathbf{R}$ for the set of real numbers. $\mathbf{R}^+$ is defined as $\{x \in \mathbf{R} : x \geq 0\}$. For any $x \in \mathbf{R}^+$ we write $\lceil x \rceil$ for the least $n \in \mathbf{N}$ with $n \geq x$.

$\{0,1\}^*$ denotes the set of all binary strings of finite length.

For any set S we write $\exists x \in S(\ldots)$ instead of $\exists x(x \in S$ and $\ldots)$, and $\forall x \in S(\ldots)$

instead of $\forall x(x \in S \rightarrow \ldots)$.

For two functions $f, g : \mathbf{N} \rightarrow \mathbf{N}$ we write $f = O(g)$ if there exists some constant c such that $f(n) \leq c \cdot g(n)$ for all except possibly finitely many $n \in \mathbf{N}$.

Structure of this Article

In section 2 we specify our basic assumptions about the response- and threshold-functions of an SNN, and we construct SNN's that can simulate in real-time arbitrary threshold-circuits and Turing machines. In section 3 we relate the computational power of SNN's for real-valued inputs to a specific type of random acces machine. In section 4 we discuss variations of the preceding constructions for related models of spiking neurons, and in section 5 we outline some conclusions from the results in this article.

2 Simulation of Threshold Circuits and Turing Machines by Networks of Spiking Neurons

In order to carry out computations on an SNN, *some* assumptions have to be made about the structure of the response- and threshold-functions of its neurons. It is obvious that for example neurons with response-function $\varepsilon_{u,v}$ such that $\varepsilon_{u,v}(s) = 0$ for all $s \geq 0$ cannot carry out *any* computation. We will specify in the following a set of **basic assumptions**, which suffice for the constructions in this article. Some variations of these conditions will be discussed in section 4.

We assume that there exist some arbitrary given constants $\Delta_{\min}, \Delta_{\max} \in \mathbf{R}$ with $0 \leq \Delta_{\min} < \Delta_{\max}$ so that we can choose for each "synapse" $\langle u, v \rangle \in E$ an individual "*delay*" $\Delta_{u,v} \in [\Delta_{\min}, \Delta_{\max}]$ with $\varepsilon_{u,v}(x) = 0$ for all $x \in [0, \Delta_{u,v}]$. This parameter $\Delta_{u,v}$ corresponds in biology to the time-span between the firing of the presynaptic neuron u and the moment when its effect reaches the trigger zone (axon hillock) of the postsynaptic neuron v. This time-span is known to vary for individual neurons in biological neural systems, depending on the type of synapse and the geometrical constellation. The constants $\Delta_{\min}$ and $\Delta_{\max}$ can be interpreted as biological constraints on the possible lengths of such time-spans. No requirements about $\Delta_{\min}$ and $\Delta_{\max}$ are needed for our construction, except that $\Delta_{\min} < \Delta_{\max}$.

We assume that except for their individual delays $\Delta_{u,v}$ the response-functions $\varepsilon_{u,v}$ (as well as the threshold functions Θ_v) are *stereotyped*, i.e. that their shape is determined by some general functions $\varepsilon^E, \varepsilon^I$ and Θ which do not depend on u or v. More precisely, we assume that we can decide for any pair $\langle u, v \rangle \in E$ whether $\varepsilon_{u,v}$ should represent an excitatory "*EPSP-response-function*", or an inhibitory "*IPSP-response-function*". In the EPSP-case we assume that

$$\varepsilon_{u,v}(\Delta_{u,v} + x) = \varepsilon^E(x) \quad \text{for all } x \in \mathbf{R}^+,$$

and in the IPSP-case we assume that

$$\varepsilon_{u,v}(\Delta_{u,v} + x) = \varepsilon^I(x) \quad \text{for all } x \in \mathbf{R}^+.$$

In either case we assume that

$$\varepsilon_{u,v}(x) = 0 \quad \text{for all } x \in [0, \Delta_{u,v}].$$

Furthermore we assume for all neurons $v \in V - V_{in}$ that

$$\Theta_v(x) = \Theta(x) \quad \text{for all } x \in \mathbf{R}^+.$$

We assume that the three functions $\varepsilon^E : \mathbf{R}^+ \to \mathbf{R}^+$, $\varepsilon^I : \mathbf{R}^+ \to \{x \in \mathbf{R} : x \leq 0\}$, and $\Theta : \mathbf{R}^+ \to \mathbf{R}^+ \cup \{\infty\}$ are some *arbitrary* functions with the following properties: There exist some arbitrary strictly positive real numbers τ_{ref}, τ_{end}, σ_1, σ_2, σ_3, τ_1, τ_2, τ_3, L, s_{up}, s_{down} with $0 < \tau_{ref} < \tau_{end}$, $\sigma_1 < \sigma_2 < \sigma_3$, $\tau_1 < \tau_2 < \tau_3$ (see Figure 1 for an illustration), which satisfy the following five conditions:

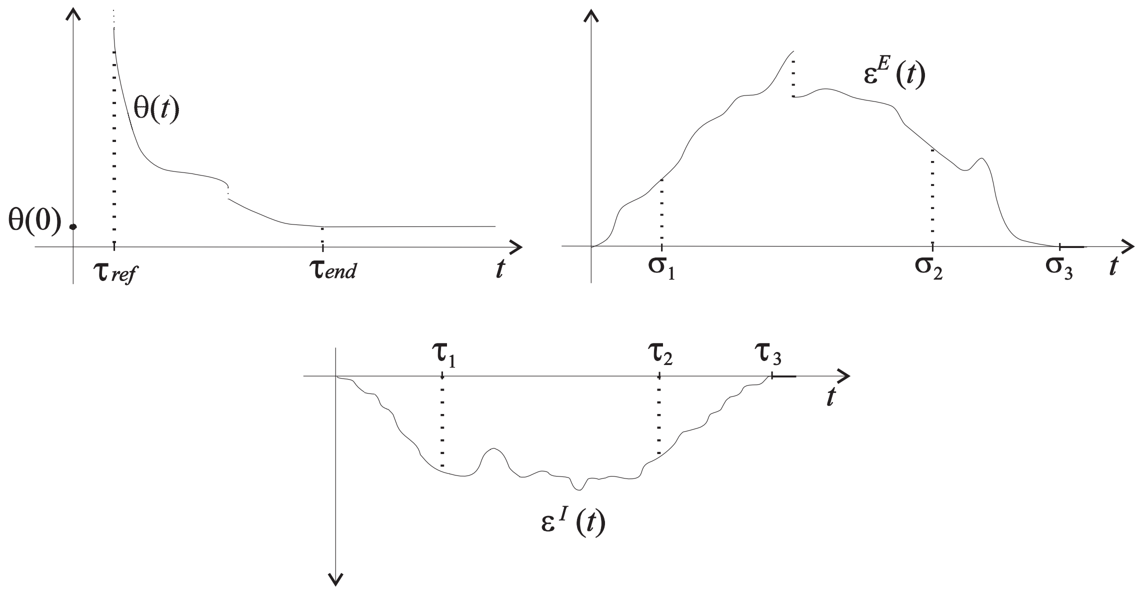

Figure 1: *Illustration of our notation for the basic assumptions on* Θ, ε^E, ε^I *(the functions shown are quite arbitrary and complicated, but nevertheless they satisfy our basic assumptions).*

(1) $\Theta(x) \geq \Theta(0) > 0$ *for all* $x \in \boldsymbol{R}^+$, $\Theta(x) = \infty$ *for all* $x \in (0, \tau_{ref})$, *and* $\Theta(x) = \Theta(0) < \infty$ *for all* $x \in [\tau_{end}, \infty)$

(2) $\varepsilon^E(0) = \varepsilon^E(x) = 0$ *for all* $x \in [\sigma_3, \infty)$, *and there exists some* $\varepsilon_{\max} \in \boldsymbol{R}^+$ *so that* $\exists x \in \boldsymbol{R}^+(\varepsilon^E(x) = \varepsilon_{\max})$ *and* $\forall\, y \in \boldsymbol{R}^+(\varepsilon^E(y) \leq \varepsilon_{\max})$

(3) $\varepsilon^E(\sigma_1 + z) = \varepsilon^E(\sigma_1) + s_{up} \cdot z$ *for all* $z \in [-L, L]$

(4) $\varepsilon^E(\sigma_2 + z) = \varepsilon^E(\sigma_2) - s_{down} \cdot z$ *for all* $z \in [-L, L]$

(5) $\varepsilon^I(0) = \varepsilon^I(x) = 0$ *for all* $x \in [\tau_3, \infty)$, $\varepsilon^I(x) < 0$ *for all* $x \in (0, \tau_3)$, ε^I *is non-increasing in* $[0, \tau_1]$ *and non-decreasing in* $[\tau_2, \tau_3]$.

We assume in addition that $\Theta(0)$, $\varepsilon^E(\sigma_1)$, $\varepsilon^E(\sigma_2)$, s_{up}, $s_{down} \in \boldsymbol{Q}$.

It should be pointed out that no conditions about the smoothness, the continuity, or the number of extrema of the functions Θ, ε^E, ε^I are made in the preceding basic assumptions. However if one demands in addition that ε^E is piecewise linear and continuous, then the conditions (3) and (4) become redundant. The assumption that $\Theta(0)$, $\varepsilon^E(\sigma_1)$, $\varepsilon^E(\sigma_2)$, s_{up}, s_{down} are rationals will only be needed to ensure that certain weights can be chosen to be *rationals* (see subsection 2.9).

Examples of mathematically particularly simple (piecewise linear) functions ε^E, ε^I and Θ which satisfy all of the above conditions are exhibited in Figure 2.

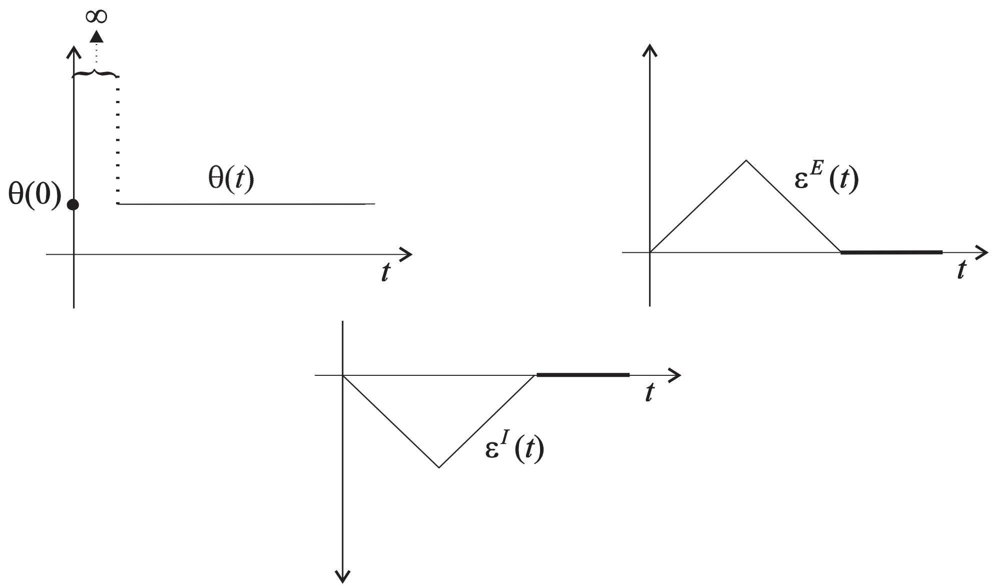

Figure 2: *Examples for mathematically very simple functions* Θ, ε^E, *and* ε^I *which satisfy the basic assumptions.*

The subsequent construction shows that neurons with the very simple response- and threshold-functions from Figure 2 can in principle be used in order to build an artificial neural network with some finite number $n_{\mathcal{U}}$ of spiking neurons that can simulate in real time any other digital computer (even computers that employ many more than $n_{\mathcal{U}}$ memory cells or computational units).

We have formulated the preceding basic assumptions on the response- and threshold-functions in a rather general fashion in order to make sure that they can in principle be satisfied by a wide range of EPSP's, IPSP's and threshold-functions that have been observed in a number of biological neural systems (see e.g. Figure 3).

The currently available findings about biological neural systems (see e.g. Kandel et al., 1991, and the discussions in Valiant, 1994) indicate that in general a single EPSP alone cannot cause a neuron to fire. In fact, it is commonly reported that 50 to 100 EPSP have to arrive within a short time-span at a neuron in order to trigger its firing. These reports indicate that the weights $w_{u,v}$ in our model should be assumed to be relatively small, since they cannot amplify a single EPSP to yield an

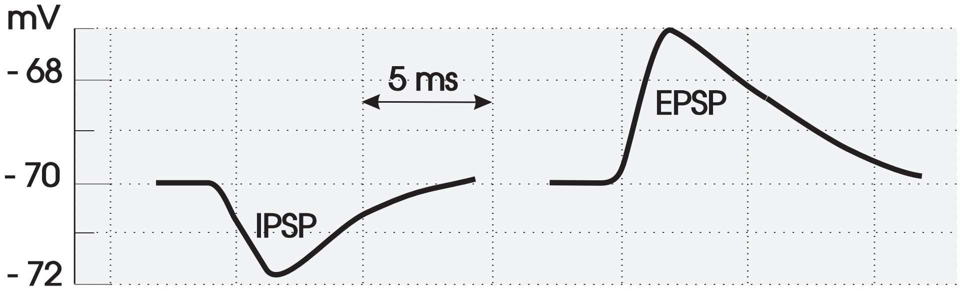

Figure 3: *Inhibitory and excitatory postsynaptic potentials at a biological neuron. (After Schmidt (1978). Fundamentals of Neurophysiology. Berlin: Springer-Verlag.)*

arbitrarily high potential P_v. Hence for the sake of biological plausibility one should assume that the values of all weights $w_{u,v}$ in an SNN belong to some bounded interval $[0, w_{\max}]$. For simplicity we assume in the following that $w_{\max} = 1$. This convention just amounts to a certain scaling of the values of the response-functions in relation to the threshold-functions. In any version of this model where a single neuron is not able to cause the firing of another neuron, one necessarily has to assume that each input spike is simultaneously received by *several* neurons (since otherwise it cannot have any effect).

In spite of this convention we will occassionally assign much larger values to certain weights $w_{u,v}$. We will then (silently) assume that u does in fact represent an *assembly* of $\lceil w_{u,v} \rceil$ neurons that all fire concurrently ($\lceil x \rceil$ is defined as the least natural number $\geq x$). Furthermore we assume in those situations that all edges from neurons in this assembly to neuron v have the same delay, and the same weight $\frac{w_{u,v}}{\lceil w_{u,v} \rceil} \in [0, 1]$. The main difference between this type of construction and a construction with arbitrarily large weights is that in our set-up the (virtual) use of large weights blows up the number of neurons that are needed.

Theorem 2.1 *If the response- and threshold-functions of the neurons satisfy the previously described basic assumptions, then one can build from such neurons for any given $d \in \boldsymbol{N}$ an SNN $\mathcal{N}_{TM}(d)$ of finite size that can simulate with a suitable assignment of rational values from $[0, 1]$ to its weights any Turing machine with at most d tapes in real-time.*

Furthermore $\mathcal{N}_{TM}(2)$ can compute $\underline{\text{any}}$ function $F : \{0, 1\}^* \rightarrow \{0, 1\}^*$ with a suitable assignment of real values from $[0, 1]$ to its weights.

The **proof** of Theorem 2.1 is rather complex. Therefore we have divided it into subsections 2.1 to 2.10, which are devoted to different aspects respectively modules of the construction. Several of these modules are also useful for other constructions. The global construction of $\mathcal{N}_{TM}(d)$ with the properties claimed in Theorem 2.1 is described in the last subsection 2.10.

We will discuss afterwards in section 4 some methods for alternative constructions of $\mathcal{N}_{TM}(d)$ that are based on different assumptions about response- and threshold-functions.

2.1 Conditions on the Neurons

We assume that we can decide for any pair $\langle u, v\rangle$ of neurons whether there should be a "synapse" between both neurons (i.e. $\langle u, v\rangle \in E$). Selfreferential edges of the form $\langle u, u\rangle$ will not be needed. In this proof the weights $w_{u,v}$ on edges $\langle u, v\rangle$ are always assumed to be time-invariant, and they are only assigned values from $[0,1]$. We assume that the response- and threshold-functions satisfy the previously described basic assumptions.

2.2 Delay- and Inhibition Modules

We will construct in this subsection two very simple modules that will be used frequently (and often silently) in the subsequent constructions. From the general point of view the existence of these two modules shows that our very weak assumptions about $\Delta_{\min}$ and $\Delta_{\max}$ (we have only required that $0 \leq \Delta_{\min} < \Delta_{\max}$) as well as our very weak assumptions about the shape of ε^I in condition (5) are in fact sufficient to create in an SNN arbitrarily long delays, and arbitrarily fast appearing or arbitrarily fast disappearing inhibitions of arbitrarily long duration.

A "*delay-module*" is simply a chain $u_1, \ldots, u_{k+1}$ of neurons so that $\langle u_i, u_{i+1}\rangle \in E$, $\varepsilon_{u_i,u_{i+1}}$ is an EPSP-response-function, and $w_{u_i,u_{i+1}} := \Theta(0)/\varepsilon_{\max}$ for $i = 1, \ldots, k$. Since each delay $\Delta_{u_i,u_{i+1}}$ can be chosen arbitrarily from $[\Delta_{\min}, \Delta_{\max}]$, the total "delay" between the firing of u_1 and the arrival of an EPSP at u_{k+1} can be chosen to assume any value in a certain interval of length $k \cdot (\Delta_{\max} - \Delta_{\min})$. It will cause no problem that the total transmission time from u_1 to u_{k+1} grows along with k, since in the subsequent constructions time will essentially only be considered *modulo* a certain constant π_{PM}.

We next construct for any given real numbers $\delta, \lambda > 0$ and $\kappa < 0$ "*inhibition-modules*" $I_{\delta,\kappa,\lambda}$ and $I^{\delta,\kappa,\lambda}$. $I_{\delta,\kappa,\lambda}$ can be used to transmit to any desired neuron v a volley of IPSP's that sum up to a potential which changes from its initial value 0 to some value $\leq \kappa$ within a time interval of length δ, and then maintains a value $\leq \kappa$ for at least the following time interval of length λ. $I_{\delta,\kappa,\lambda}$ consists of a neuron u that transmits EPSP's simultaneaously to several "relay-neurons" $v_1, \ldots, v_l$, which are triggered by this EPSP to send an IPSP to some given neuron v. If l and the delays between the neurons are chosen appropriately (as a function of $\delta, \kappa, \lambda, \varepsilon^I(\delta)$ and the parameter τ_1, this module will have the desired effect on neuron v.

Dually, one can also build for any $\delta, \lambda > 0$ and $\kappa < 0$ an inhibition module $I^{\delta,\kappa,\lambda}$ that sends IPSP's to any specified neuron v whose sum stays $\leq \kappa$ for a time interval of length $\geq \lambda$, and then returns to 0 within a time interval of length $\leq \delta$. Here we exploit that according to condition (5) the function $\varepsilon^I(x)$ is non-decreasing and strictly negative for $x \in [\tau_2, \tau_3)$.

2.3 Oscillators

Consider subgraphs of an SNN of the following structure:

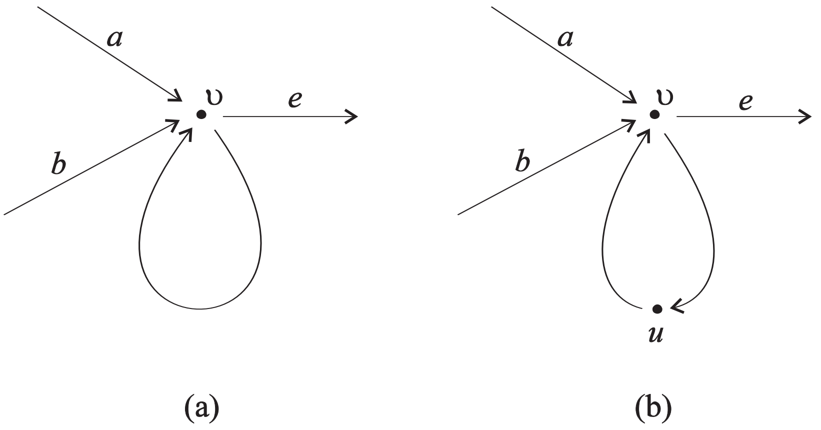

Figure 4: *Graph structure of an oscillator consisting of one neuron (a), respectively two neurons (b).*

Both types of subgraphs can be used to build an oscillator. The first one is somewhat simpler, but we will not use it in our construction since it would require a selfreferential edge $\langle v, v \rangle \in E$.

In the second type of oscillator (Figure 4, (b)) we assume that $w_{u,v}, \quad w_{v,u} \geq \Theta(0)/\varepsilon_{\max}$, and that both $\varepsilon_{u,v}$ and $\varepsilon_{v,u}$ are EPSP-response-functions. Thus after an initial EPSP through edge a both neurons will fire periodically. More precisely, v will fire at times $t_0 + i \cdot \pi$ for $i = 1, 2, \ldots$, until it is halted by an IPSP through edge b. We refer to π as the *oscillation-period* of this oscillator.

We will distinguish one such oscillator as the "*pacemaker*" for the constructed SNN, which we denote by PM. We write π_{PM} for its oscillation period. We assume that the oscillation of PM is started at "time 0" by the first input spike to the SNN, and that it continues without interruption throughout the computation of the SNN. PM emits EPSP's through edge e, which will then be broadcast as a timing-standard throughout the SNN. We will say in the following that some other neuron v in the

SNN fires "*at unit time*" or "*synchronously*" if the considered firing of v occurs at a time point t of the form $i \cdot \pi_{\text{PM}}$ for some $i \in \mathbf{N}$.

In $\mathcal{N}_{TM}(d)$ we will use oscillators in two ways as storage devices. First we use them as "registers" for storing a bit (via their two states dormant/oscillating), for example in the control of $\mathcal{N}_{TM}(d)$. Secondly we use oscillators O with oscillation period π_{PM} to store arbitrary numbers $\varphi \in [0, \pi_{\text{PM}}]$ via their *phase difference* to PM (i.e. neuron v of oscillator O fires at time points of the form $i \cdot \pi_{\text{PM}} + \varphi$ with $i \in \mathbf{N}$). In this way oscillators can for example store the time difference between two input-spikes to the SNN, and the program respectively tape content of a simulated Turing machine.

2.4 Synchronization Modules

A characteristic feature of a computation on a feedforward boolean circuit of the usual type is that the *timing* of its computation steps is *independent* of the *values* of the bits that occur in the computation. For example, the timing of the output signal of an OR-gate does not depend on the values of its input bits. This feature is very useful, since with its help one can arrange that all input bits for boolean gates on higher levels of the circuit arrive simultaneously, and therefore it allows us to build complex circuits from simple modules.

If one wants to carry out computations on an SNN with single spikes, one would like to interpret the firing of a neuron at a certain time as the bit "1" and non-firing as "0". Thus one might for example want to simulate an OR-gate by a neuron v that fires whenever it receives at least one EPSP. However when that neuron v receives *two* EPSP's simultaneously (corresponding to *two* input bits being 1) it would in general fire slightly *earlier* than in a situation where it receives just a *single* EPSP. This effect is a consequence of having EPSP-response-functions $\varepsilon_{u,v}(x)$ that are not piecewise constant. In addition, if v has already fired just before, then the fact that $\Theta(x)$ is in general not piecewise constant also contributes to this effect. Unfortunately this effect makes it impossible to simulate on an SNN in a straightforward manner a multi-layer boolean circuit (where the bit "1" is signaled by a spike, and "0" by the absence of a spike at the corresponding time): the input "bits" for neurons that simulate boolean gates on higher layers of the circuit will in general not arrive at the same time. Furthermore it is *not* possible to correct this problem by employing delay modules of the type that we had constructed in subsection 2.2, since the required length of the delays depends on the current values of the input bits.

We will solve this problem with the help of the here constructed *synchronization module*. In fact, we will show in the next subsection that with the help of this module an SNN suddenly gains the full computational power of a boolean feedforward *threshold circuit*, and therefore is able to carry out within a small number of "cycles" substantially more complex computations than a regular boolean circuit.

On first sight it appears to be impossible to build a synchronization module without postulating the existence of an EPSP-response-function that has segments of length $\geq \pi_{PM}$ where it is constant, or increases respectively decreases linearly. However the following "double-negation trick" allows us to build a synchronization module without any additional assumptions.

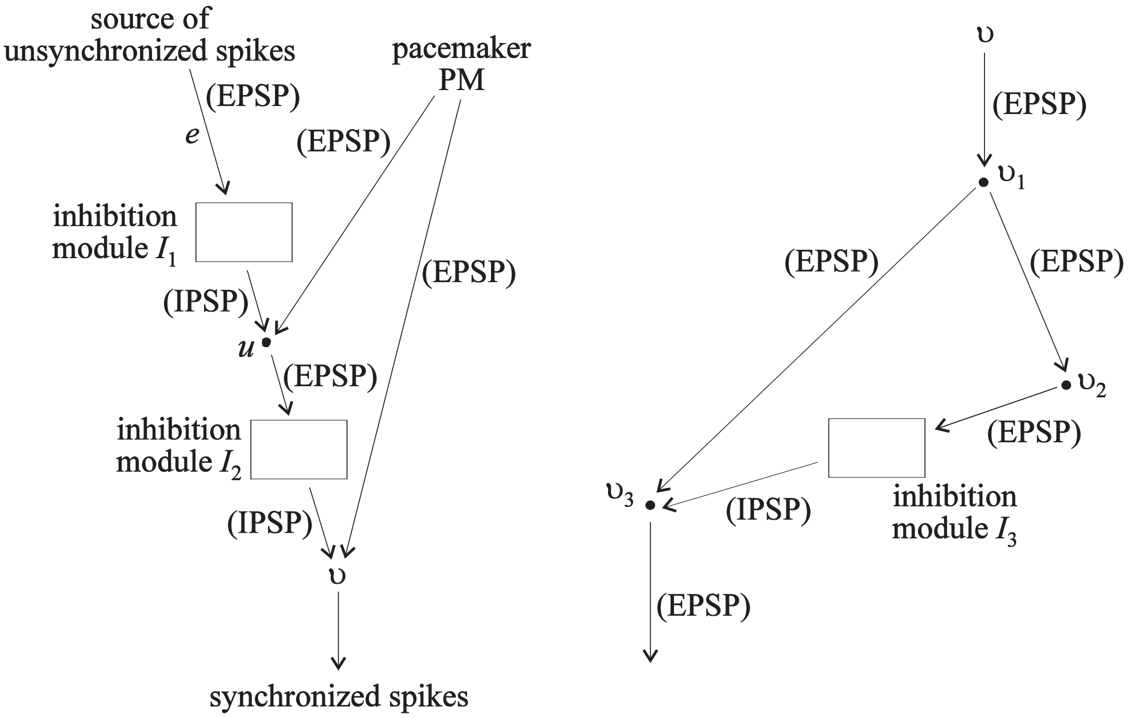

Figure 5: *Structure of a synchronization module.*

Consider the graph of an SNN on the left hand side of Figure 5. We arrange that as long as no EPSP is transmitted through its "input edge" e, the neuron u fires regularly with period π_{PM} as a result of EPSP's from the pacemaker PM. These EPSP's induce the inhibition module I_2 to send IPSP's to neuron v that "cancel out" the EPSP's that arrive at v directly from PM. Therefore in the absence of an input through edge e this neuron v does not fire.

Assume now that at some arbitrary time point an (unsynchronized) EPSP arrives through edge e. This EPSP triggers the inhibition module I_1, which then sends out IPSP's that prevent neuron u from firing for a time interval of some fixed length $> \pi_{\mathrm{PM}}$. Therefore at least one of the EPSP's that arrive at neuron v from PM is not cancelled out by IPSP's from the inhibition module I_2, and neuron v emits at least one synchronized spike (i.e. v fires at least once, and with a proper choice of delays only at unit times of the form $i \cdot \pi_{\mathrm{PM}}$ with $i \in \mathbf{N}$).

A closer look shows that the mechanism of this module is in fact a bit more delicate. It can in principle happen that at neuron u the beginning or the end of a negative potential from I_1 coincides with an EPSP from PM in such a way that it leads to a small shift ρ in some firing time of u (besides cancelling other firings of u). This could shift the time-interval of the activity of I_2 by a certain amount ρ. One has to make sure that this shift cannot lead to a competition at neuron v between the negative potential from I_2 and the EPSP from PM that results in an *unsynchronized* firing of v. One can solve this technical problem by designing I_1 and I_2 so that their output is the superposition of the output of a module $I_{\delta,\kappa,\lambda}$ *and* of a module $I^{\delta,\kappa,\lambda}$. In this way one can achieve that their strongly negative output-potential (of value $\leq \kappa$ both builds up and disappears at neuron v within time-intervals of length δ. This parameter δ provides then an upper bound for the length ρ of the possible time-shifts of these negative potentials. By choosing δ sufficiently small (and by arranging the lengths and delays of these inhibitions appropriately) one can achieve that for *any* arrival time of an input spike through edge e and for any EPSP from PM the resulting inhibition from I_2 either cancels the corresponding firing of v, or it lets v fire without shifting its firing time (cancelling some other firings of v instead). For that purpose one chooses the weight $w \in [0,1]$ on the edge from PM to v so that the resulting function $w \cdot \varepsilon^E$ crosses $\Theta(0)$ while it is in the middle of its linearly increasing segment (see condition (3) of our basic assumptions).

The timing of this synchronization module can be specified with more precision as soon as one selects concrete response- and threshold-functions that satisfy our basic assumptions. However the preceding analysis shows that it will do its job in any case. One should keep in mind that our basic assumptions are relatively weak. For example they do not even prescribe the relationships between the sizes of the parameters σ_3, τ_3, and τ_{end} that denote the lengths of the non-trivial segments of the response- and threshold-functions.

It turns out that the previously described module may output not just one, but a larger finite number of synchronized spikes as a result of one unsynchronized input spike. This effect causes no serious problem in our subsequent applications of this module (and it might occasionally be helpful for speeding up a computation), but it is easier to verify a construction if this module never outputs more than *one* synchronized spike for each input spike. This additional requirement can be satisfied by adding after neuron v a device with three neurons v_1, v_2, v_3 as indicated in the right hand side of Figure 5. With suitably chosen delays and parameters for its inhibition module I_3, this device removes all except the first spike from any sequence of successive synchronized spikes. It lets the first one of these spikes emerge from neuron v_3 as a *single synchronized* output spike.

2.5 Simulation of Boolean Threshold Circuits by SNN's

If one just wants to simulate in a straightforward manner the control of a Turing machine on an SNN, one can reserve one neuron for each possible state of the control, and simulate state-transitions with the help of neurons that simulate boolean AND- and OR-gates. However Horne and Hush, 1993, have pointed out that much fewer neurons are needed if one simulates the control with the help of a boolean feedforward threshold circuit with gates of unbounded fan-in (see subsection 2.8). In addition, the ability of SNN's to simulate threshold circuits in an efficient manner is of substantial interest for various other reasons (see Corollary 2.4 and the lower bound for the VC-dimension of SNN's in Maass, 1994b). Therefore we describe here rightaway the simulation of a *threshold circuit* on an SNN, rather than considering first the simulation of the special case of a boolean circuit with gates of bounded fan-in (which would suffice for the proof of Theorem 2.1).

A feedforward boolean threshold circuit (*threshold circuit* for short) consists of a directed acyclic graph with nodes of arbitrary fan-in, that correspond to linear threshold gates (*threshold gates* for short) with arbitrary weights. A threshold gate with fan-in m computes a *threshold function* of the form

$$\{0,1\}^m \ni \langle x_1, \ldots, x_m \rangle \mapsto T^{\underline{\alpha}}(x_1, \ldots, x_m) = \begin{cases} 1, & \text{if } \sum\limits_{i=1}^{m} \alpha_i \cdot x_i \geq \alpha_0 \\ 0, & \text{otherwise} \end{cases}$$

with arbitrary parameters $\alpha_0, \ldots, \alpha_m \in \mathbf{R}$ (or equivalently: $\alpha_0, \ldots, \alpha_m \in \mathbf{Z}$) .

It is obvious that the common boolean operations AND, OR, NOT are special cases of threshold functions. Therefore the common types of feedforward boolean circuits (even with AND's and OR's of arbitrarily large fan-in) are special cases of threshold circuits. Hence, since every boolean function can be defined by a boolean formula in disjunctive normal form (see e.g. Lewis and Papadimitriou, 1981) it is clear that *every* boolean function can be computed by a threshold circuit of depth 2 (i.e. with one "hidden" layer).

There are several different possibilities for simulating a threshold circuit on an SNN, providing subtle tradeoffs between the amount of demands imposed on the response-functions, the noise-robustness of the construction, and the number of neurons needed for the simulation. We describe here one simple construction based on our basic assumptions, and we will indicate a variation in section 4.

Consider first a "monotone" threshold function, i.e. a threshold function $T^{\underline{\alpha}}$ with $\alpha_i \geq 0$ for all "weights" $\alpha_1, \ldots, \alpha_m$. If $\alpha_0 \leq 0$ then $T^{\underline{\alpha}}$ always outputs "1", and is therefore superfluous. Hence we may assume that $\alpha_0 > 0$.

By condition (2) each EPSP-response-function $\varepsilon_{u,v}$ has some maximal value $\varepsilon_{\max} > 0$ which does not depend on u or v. We employ for the computation of $T^{\underline{\alpha}}$ on an SNN $m+1$ neurons $u_1, \ldots, u_m$ and v with $\{u : \langle u, v \rangle \in E\} = \{u_1, \ldots, u_m\}$. We assume that all response-functions $\varepsilon_{u_i,v}$ are EPSP's and that the weights $w_{u_i,v}$ are chosen so that $w_{u_i,v} \cdot \varepsilon_{\max} = \alpha_i \cdot \frac{\Theta(0)}{\alpha_0}$ (slightly larger values should be chosen if one has to deal with imprecision). Furthermore we assume that the "delays" $\Delta_{u_i,v}$ are chosen to be the same for $i = 1, \ldots, m$. Consider then some arbitrary set $S \subseteq \{1, \ldots, m\}$. Assume that the neurons u_i with $i \in S$ fire simultaneously at some time t_0, that the neurons u_i with $i \in \{1,\ldots,m\} - S$ never fire in the time interval $[t_0 - \sigma_3, \quad t_0 + \sigma_3]$, and that neuron v did not fire in the time interval $[t_0 + \Delta_{u_1,v} - \tau_{end}, \quad t_0 + \Delta_{u_1,v}]$. Then v fires at some point in the time interval $(t_0 + \Delta_{u_1,v}, \quad t_0 + \Delta_{u_1,v} + \sigma_3)$ if and only if $\sum\limits_{i \in S} w_{u_i,v} \cdot \varepsilon_{\max} \geq \Theta(0)$. The latter inequality is equivalent to $\sum\limits_{i \in S} \alpha_i \cdot \frac{\Theta(0)}{\alpha_0} \geq \Theta(0)$, hence to $\sum\limits_{i \in S} \alpha_i \geq \alpha_0$. Thus we have constructed a module of an SNN that computes an arbitrary monotone boolean threshold function $T^{\underline{\alpha}}$.

This module has the disadvantage that its proper functioning is only guaranteed if all u_i with $i \in S$ fire at a *common* time t_0. On the other hand the firing time of v depends not only on t_0, but also on S (i.e. on its "input bits"). In general a larger set S gives rise to a slightly earlier firing time of v (because the function ε^E does not jump immediately from 0 to $\varepsilon_{\max}$). Obviously these two facts together cause problems if one wants to use compositions of the previously constructed module in order to simulate a multi-layer monotone threshold circuit (i.e. a threshold circuit where all gates compute *monotone* threshold functions). Therefore one has to use synchronization modules between any two layers of modules in order to simulate on an SNN a *monotone threshold circuit.*

We will now describe the simulation of an *arbitrary* threshold circuit C, where threshold functions $T^{\underline{\alpha}}$ with "weights" α_i of arbitrary sign are computed by gates of C. It is well-known (see Hajnal et al., 1993) that such a circuit C can be simulated by a *monotone* threshold circuit C_{mon} of the same depth, provided that C_{mon} receives for each boolean input variable x_i also its negation $1 - x_i$. Proceeding from the input layer to the ouput layer one can then replace each threshold gate g of C by two gates that both compute *monotone* threshold functions: one of them provides the same output as g, and the other one provides the negation of that output.

Thus in order to simulate C on an SNN, one needs in addition to the preceding construction a preprocessing device that computes the negation $1 - x$ for each input bit $x \in \{0, 1\}$ under the here considered bit-encoding (where "$x = 1$" is encoded by a firing of a neuron u at a certain time t, and "$x = 0$" by the non-firing of u within a certain time interval around t). For that purpose one connects u to an inhibition module whose outputs cancels out an EPSP from PM at another neuron u' (similarly as in subsection 2.4). Then u' will fire if and only if it is not inhibited via a firing of u, hence u' computes "$1 - x$".

2.6 Modules for Comparisons and Multiplication of Phases with Arbitrary Constants

We will construct in this subsection a module for an SNN that can compare the phase-difference φ of an oscillator O with some given constant α (COMPARE($\geq \alpha$)), and a module that can multiply φ with some given constant β (MULTIPLY(β)). Such modules (more precisely: modules for the operation COMPARE($\geq 2^{-1-c}$) for a certain constant c, as well as modules for MULTIPLY(2) and MULTIPLY(1/2)) will be needed in the next subsection in order to simulate a stack on an SNN.

Let $\alpha \in [0, L/2]$ be some arbitrary real constant. We construct a module which can decide whether the phase-difference $\varphi \in [0, L/2]$ between PM and some oscillator O with oscillation period π_{PM} is $\geq \alpha$. More precisely, this module for the operation COMPARE($\geq \alpha$) will send out a spike within some time-interval of some given length $2D$ if and only if $\varphi \geq \alpha$. Consider neurons u_1, u_2 and v with $\langle u_i, v\rangle \in E$ for $i = 1, 2$. Assume that u_1 is induced to fire at a certain time t_1 by a spike from the pacemaker PM. Furthermore assume that u_2 is induced to fire at a certain time t_2 by a spike from the oscillator O. Finally we assume that the delays $\Delta_{u_1,v}$ and $\Delta_{u_2,v}$ have been chosen so that in the case $\varphi = \alpha$ one has for $\tilde{t}_i := t_i + \Delta_{u_i,v}$ that there exists some $t^* \geq \max(\tilde{t}_1, \tilde{t}_2)$ so that $t^* - \tilde{t}_1 = \sigma_1$ and $t^* - \tilde{t}_2 = \sigma_2$. We choose weights $w_{u_i,v} > 0$ so that $w_{u_1,v} \cdot s_{up} = w_{u_2,v} \cdot s_{down}$ and $w_{u_1,v} \cdot \varepsilon^E(\sigma_1) + w_{u_2,v} \cdot \varepsilon^E(\sigma_2) = \Theta(0)$ (see Figure 6).

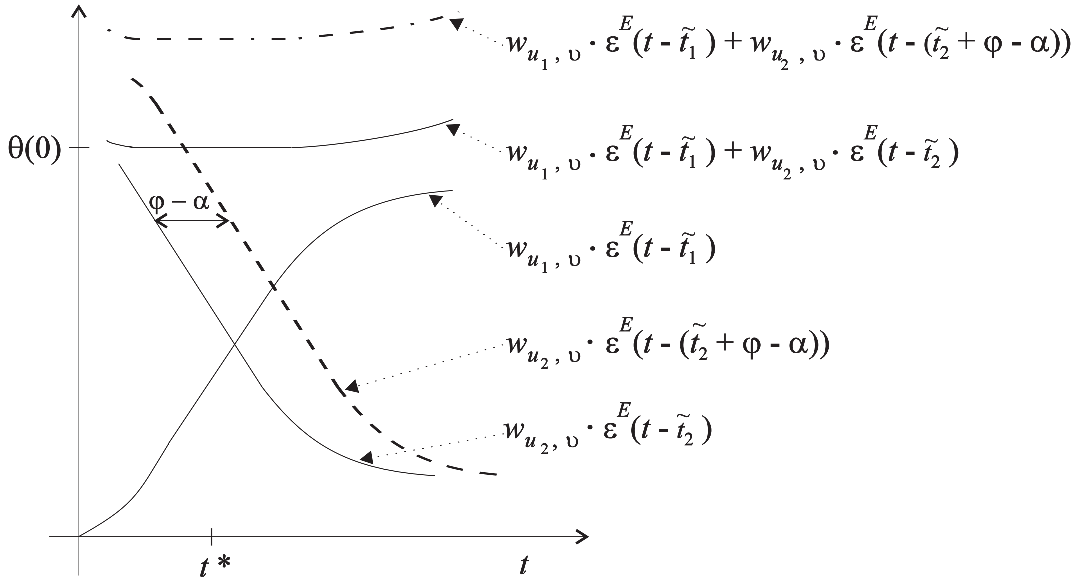

Figure 6: *Mechanism of the module for COMPARE($\geq \alpha$).*

According to our general convention at the beginning of this section we actually have to replace in the case $w_{u_i,v} > 1$ the neuron u_i by an assembly of $\lceil w_{u_i,v} \rceil$ neurons with weights from $[0, 1]$ on their edges to v. However for the sake of simplicity we will ignore this trivial complication in the following.

We arrange that for an arbitrarily given parameter $D > 0$ inhibition modules $I_{\delta,\kappa,\lambda}$ and $I^{\tilde{\delta},\tilde{\kappa},\tilde{\lambda}}$ (with suitable values of their parameters) are triggered by spikes from PM to send IPSP's to v so that v is not able to fire within the time intervals $[t^* - L/2 - D,\ t^* - L/2)$ and $(t^* + L/2,\ \ t^* + L/2 + D]$ even if the firing time t_2 of neuron u_2 is arbitrarily shifted, but so that these inhibition modules have no effect on the potential P_v at neuron v during the time interval $[t^* - L/4,\ \ t^* + L/4]$.

Consider now what happens if the phase-difference φ of the oscillator O is not fixed at $\varphi = \alpha$, but assumes any value in $[0, L/2]$. Then by choice of the parameters $w_{u_1,v}, w_{u_2,v}$, and t^*, and by the conditions (3) and (4) of our basic assumptions, the sum of the EPSP's from u_1 and u_2 at neuron v has in any case a constant value within the time interval $[t^* - L/2, \ \ t^* + L/2]$. Furthermore this constant value is $\geq \Theta(0)$ if and only if $\varphi \geq \alpha$. Hence the neuron v will fire within the time interval $[t^* - L/2, \ t^* + L/2]$ if and only if $\varphi \geq \alpha$. Furthermore by the choice of the inhibition modules the neuron v fires within the time interval $[t^* - L/2, \ t^* + L/2]$ if and only if it fires within the time interval $[t^* - D, \ t^* + D]$.

We now assume that some arbitrary real number $\beta > 0$ is given, and we construct a module that carries out the operation MULTIPLY(β). This module also consists of neurons u_1, u_2, v with $\langle u_i, v \rangle \in E$ for $i = 1, 2$ so that u_1 is triggered to fire at some time t_1 by a spike from the pacemaker PM, and u_2 is triggered to fire at some time t_2 by a spike from an oscillator O which has oscillation period π_{PM} and some phase-difference $\varphi \in [0, \min(L/2, L/2\beta)]$ to PM. We want to achieve that for any value $\varphi \in [0, \min(L/2, L/2\beta)]$ of this phase-difference the "output-neuron" v of this module fires at a time $t + \beta \cdot \varphi$, where t does not depend on φ.

The construction of the module for the operation MULTIPLY(β) is slightly different for the two cases $\beta > 1$ and $\beta \in (0, 1)$. We consider first the case $\beta > 1$. Assume for the moment that the phase-difference $\varphi \in [0, L/2\beta]$ between O and PM has value 0, and choose delays $\Delta_{u_i,v}$ so that there exists for $\tilde{t}_i := t_i + \Delta_{u_i,v}$ some $t^* \geq \max(\tilde{t}_1, \tilde{t}_2)$ with $t^* - \tilde{t}_1 = \sigma_2$ and $t^* - \tilde{t}_2 = \sigma_1$. Furthermore we choose weights $w_{u_i,v} > 0$ so that

$$\text{(a)} \qquad w_{u_1,v} \cdot \varepsilon^E(t^* - \tilde{t}_1) + w_{u_2,v} \cdot \varepsilon^E(t^* - \tilde{t}_2) \ = \ \Theta(0) \quad \text{and}$$

$$\text{(b)} \qquad \beta \ = \ \frac{w_{u_2,v} \cdot s_{up}}{w_{u_2,v} \cdot s_{up} - w_{u_1,v} \cdot s_{down}} \ .$$

Since $\beta > 1$, the equation (b) implies that $0 < w_{u_1,v} \cdot s_{down} < w_{u_2,v} \cdot s_{up}$. Hence we have

$$\text{(c)} \qquad w_{u_1,v} \cdot \varepsilon^E(t^* - \tilde{t}_1 + z) + w_{u_2,v} \cdot \varepsilon^E(t^* - \tilde{t}_2 + z) < \Theta(0) \quad \text{for all } z \in [-L, 0).$$

We would like to arrange that v does not fire during the time interval $[t^* - \tau_{end}, \ \ t^*)$, where τ_{end} has the property that $\Theta(x) = \Theta(0)$ for all $x \in [\tau_{end}, \ \ \infty)$ (according to condition (1)). Furthermore we would like to make sure that this property holds even if the firing of u_2 is delayed by some arbitrary amount $\varphi \in [0, L/2\beta]$. However even if one assumes that only the considered EPSP's from u_1 and u_2 are influencing $P_v(t)$, this assumption only allows us to derive this fact with the help of (c) for the interval $[t^* - L/2, \ \ t^*)$, since we did not make more detailed assumptions about the shape of the function ε^E. Therefore we arrange that at a suitable time an inhibition module $I^{L/2,\kappa,\tau_{end}}$ sends IPSP's to v, which make it impossible that v fires during the time interval $[t^* - \tau_{end}, \ \ t^* - L/2)$ (no matter at what time u_2 fires), but which do not influence the potential $P_v(t)$ at times $t \geq t^*$. Furthermore we arrange that no other EPSP's or IPSP's contribute to $P_v(t)$ for $t \in [t^* - \tau_{end},\ \ t^*]$. In this way it is made impossible that v fires during the time interval $[t^* - \tau_{end},\ \ t^*)$ (even if the firing of u_2 is delayed by some $\varphi \in [0, L/2\beta]$). Therefore in the case $\varphi = 0$ our assumption (a) implies that neuron v will fire at time t^*.

We now consider what will change if the firing of u_2 at time t_2 is replaced by a slightly later firing at time $t_2 + \varphi$, whereas the firing time of u_1 and of the inhibition module remain unchanged. We will show that for any $\varphi \in (0, L/2\beta]$ this delay will cause a somewhat delayed firing of v (see Figure 7). Consider the time point t_φ

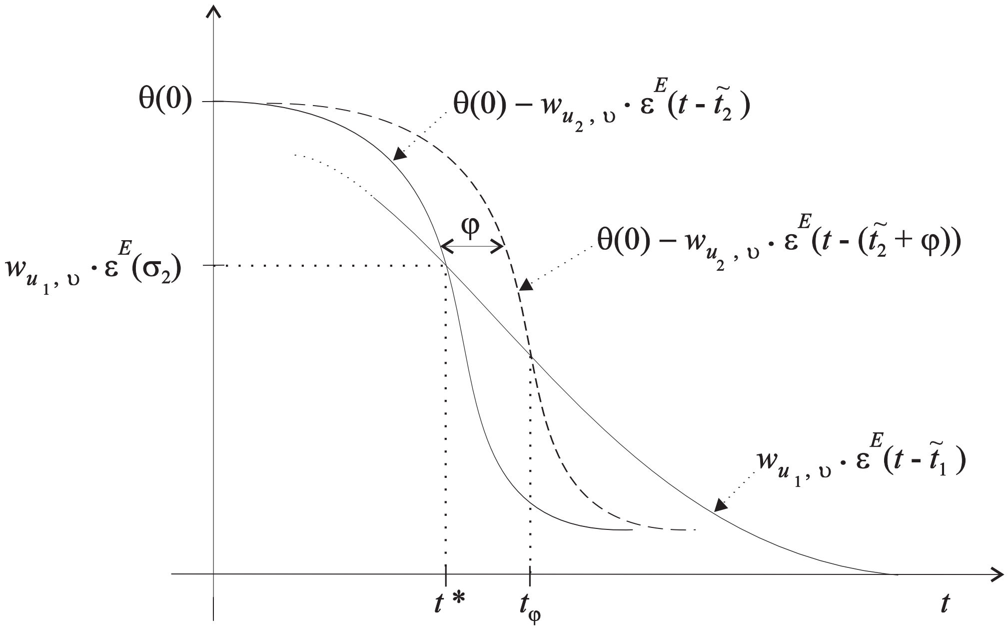

Figure 7: *Multiplication of a phase φ with $\beta > 1$ (i.e. $t_\varphi - t^* = \beta \cdot \varphi$).*

which is defined by the equation

$$\text{(d)} \qquad w_{u_1,v} \cdot \varepsilon^E(t_\varphi - \tilde{t}_1) + w_{u_2,v} \cdot \varepsilon^E(t_\varphi - (\tilde{t}_2 + \varphi)) \ = \ \Theta(0).$$

By equation (a) and the conditions (3) and (4) of our basic assumptions we have for $t_\varphi - t^* \in [-L, L]$ that

$$\text{(e)} \qquad w_{u_1,v} \cdot \varepsilon^E(t_\varphi - \tilde{t}_1) \ = \ w_{u_1,v} \cdot \varepsilon^E(t^* - \tilde{t}_1) - w_{u_1,v} \cdot s_{down} \cdot (t_\varphi - t^*),$$

and for φ, t_φ with $t_\varphi - t^* - \varphi \in [-L, L]$ we have that

$$\text{(f)} \qquad w_{u_2,v} \cdot \varepsilon^E(t_\varphi - (\tilde{t}_2 + \varphi)) \ = \ w_{u_2,v} \cdot \varepsilon^E(t^* - \tilde{t}_2) + w_{u_2,v} \cdot s_{up} \cdot (t_\varphi - t^* - \varphi).$$

These two equations in conjunction with (a), (b), and (d) imply that

$$t_\varphi - t^* = \beta \cdot \varphi.$$

It is obvious that for $\varphi \in [0, L/2\beta]$ one has that $\beta \cdot \varphi, \quad \beta \cdot \varphi - \varphi \in [-L, L]$. Furthermore it is clear from our construction that v cannot fire during the time interval $[t^* - \tau_{end}, \quad t^* + \beta \cdot \varphi)$. Therefore $t_\varphi := t^* + \beta \cdot \varphi$ is in fact the firing time of v if u_2 fires at time $t_2 + \varphi$. Hence the described module carries out the operation MULTIPLY(β) in case that $\beta > 1$.

In order to carry out the operation MULTIPLY(β) for some arbitrarily given $\beta \in (0, 1)$ we just change the delay $\Delta_{u_1,v}$ in the previously described module so that $t^* - \tilde{t}_1 = \sigma_1$ (instead of $t^* - \tilde{t}_1 = \sigma_2$), see Figure 8. We choose weights $w_{u_i,v} > 0$ so that (a) holds and

$$\text{(g)} \qquad \beta \;=\; \frac{w_{u_2,v}}{w_{u_2,v} + w_{u_1,v}} \quad .$$

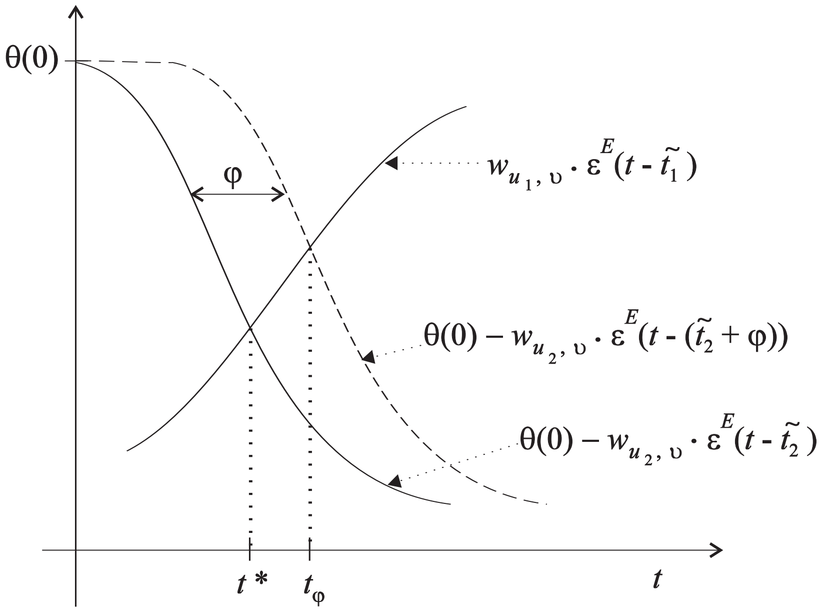

Figure 8: *Multiplication of a phase φ with $\beta \in (0, 1)$.*

As before, we consider the time point t_φ that is defined by (d). Then equation (f) holds, but instead of (e) we have

$$w_{u_1,v} \cdot \varepsilon^E(t_\varphi - \tilde{t}_1) \;=\; w_{u_1,v} \cdot \varepsilon^E(t^* - \tilde{t}_1) + w_{u_1,v} \cdot s_{up} \cdot (t_\varphi - t^*) \quad .$$

The latter two equations in conjunction with (a), (d) and (g) imply that

$$t_\varphi - t^* = \beta \cdot \varphi.$$

Hence the described module carries out the operation MULTIPLY(β) for an arbitrarily given $\beta \in (0, 1)$.

2.7 Simulation of a Stack with Unlimited Capacity by an SNN of Fixed Size

The simulation of a stack (also called pushdown store, of first in - last out list) is the most delicate part of the construction of $\mathcal{N}_{TM}(d)$, since it requires the construction of a module in which the lengths ℓ of the bit-strings $\langle b_1, \ldots, b_\ell \rangle$ that are stored and manipulated are in general much larger than the number of neurons in this module (in fact: ℓ can be arbitrarily large). Of course $\mathcal{N}_{TM}(d)$ *needs to have* a component with this property, since otherwise the SNN $\mathcal{N}_{TM}(d)$ (which will consist of a *fixed* finite number of neurons) cannot simulate the computations of Turing machines that involve tape-inscriptions of arbitrary finite length. The content $\langle b_1, \ldots, b_\ell \rangle \in \{0,1\}^*$ of a stack S (where b_1 is the symbol on top of the stack) will be stored in the form of the phase-difference

$$\varphi_S = \sum_{i=1}^{\ell} b_i \cdot 2^{-i-c}$$

of a special oscillator O_S. More precisely, we assume that O_S fires with the same oscillation period π_{PM} as the pacemaker PM, but with a delay φ_S. The parameter $c \in \mathbf{R}^+$ is some arbitrary constant which is sufficiently large so that $2^{-c} \leq \min(L/2, \pi_{\mathrm{PM}})$.

We will now describe the mechanisms for simulating the stack operations POP and PUSH on a bit string $\langle b_1, \ldots, b_\ell \rangle$ that is stored in φ_S.

The stack operation POP determines the value of the top-bit b_1, and then replaces the stack content $\langle b_1, \ldots, b_\ell \rangle$ by $\langle b_2, \ldots, b_\ell \rangle$. In an SNN one can determine the value of b_1 from φ_S by testing whether $\varphi_S \geq 2^{-1-c}$. For that purpose one employs a module that carries out the operation COMPARE($\geq 2^{-1-c}$) (see the preceding subsection).

In order to change the phase-difference φ_S from $\sum_{i=1}^{\ell} b_i \cdot 2^{-i-c}$ to $\sum_{i=1}^{\ell-1} b_{i+1} \cdot 2^{-i-c}$ one first replaces φ_S by $\sum_{i=2}^{\ell} b_i \cdot 2^{-i-c}$. This can be carried out by directing in the case $b_1 = 1$ an EPSP from O_S through a suitable delay module, by halting simultaneously the oscillation of O_S with the help of an inhibition module, and by restarting the oscillation of O_S with an EPSP from the considered delay module. Note that we can employ at this point a simple delay module as described in subsection 2.2, because in the case $b_1 = 1$ the length of the desired shift of the phase-difference does not depend on its current value.

It remains to carry out a SHIFT-LEFT operation, which replaces the phase-difference $\sum_{i=2}^{\ell} b_i \cdot 2^{-i-c}$ by $2 \cdot \sum_{i=2}^{\ell} b_i \cdot 2^{-i-c} = \sum_{i=1}^{\ell-1} b_{i+1} \cdot 2^{-i-c}$. This operation cannot be implemented by a delay-module, since it has to shift the phase-difference by an amount that depends on the values of ℓ and $b_2, \ldots, b_\ell$. Instead, we have to employ a module that carries out the operation MULTIPLY(2) (see subsection 2.6).

In order to simulate the stack operation PUSH one has to replace for a given $b_0 \in \{0,1\}$ the current phase-difference $\varphi_S = \sum_{i=1}^{\ell} b_i \cdot 2^{-i-c}$ of the oscillator O_S by $\sum_{i=1}^{\ell+1} b_{i-1} \cdot 2^{-i-c}$. Our simulation of PUSH consists of two separate parts: a SHIFT-RIGHT operation that changes the current phase difference to $\sum_{i=2}^{\ell+1} b_{i-1} \cdot 2^{-i-c}$, and a subsequent ADD($\gamma$) operation that adds $\gamma := b_0 \cdot 2^{-1-c}$ to this phase-difference. Obviously ADD(γ) can be implemented in an analogous way as the subtraction of $b_1 \cdot 2^{-1-c}$ from φ_S in the previously described simulation of POP.

Thus it just remains to simulate a SHIFT-RIGHT operation, i.e. to replace the phase difference $\varphi_S = \sum_{i=1}^{\ell} b_i \cdot 2^{-i-c}$ of size $\leq L/2$ by $\varphi_S/2 = \sum_{i=2}^{\ell+1} b_{i-1} \cdot 2^{-i-c}$. For that purpose we employ a module for the operation MULTIPLY(1/2), as constructed in the preceding subsection.

2.8 Simulation of an Arbitrary Fixed Turing Machine by an SNN

We will show in this subsection that the previously constructed modules suffice to construct for any given Turing machine M an SNN $\mathcal{N}_M$ (whose structure may depend on M) that can simulate M in real-time. According to the notion of a *real-time computation* (see section 1) we assume that the given Turing machine M processes a sequence $(\langle x(j), y(j)\rangle)_{j\in\mathbf{N}}$ with $x(j), y(j) \in \{0,1\}^*$ in real-time. We assume that the inputs $x(j)$ are presented to M on a read-only input tape, and the outputs $y(j)$ are written by M on some write-only output tape. We will assume that the simulating SNN $\mathcal{N}_M$ receives each input $x(j) \in \{0,1\}^*$ in the form of a time difference φ between two input-spikes, with $\varphi = \sum_{i=1}^{\ell} b_i \cdot 2^{-i-c}$ for $x(j) = \langle b_1, \ldots, b_\ell\rangle$. We will arrange that $\mathcal{N}_M$ delivers its outputs $y(j)$ in the same form (as a time difference between two output spikes).

It is easy to see that any Turing machine M, with any finite number d of two-way infinite read/write-tapes, can be simulated in real-time by a similar machine which has $2d$ *stacks*, but no tapes (see e.g. Hopcroft and Ullman, 1979). We will call the latter type of machine also a Turing machine. In this simulation one uses two stacks for the simulation of each tape: one stack for simulating the part of the tape that lies to the *left* of the current position of the tape-head, and another stack for simulating the part of tape to the *right* of the tape-head.

In principle it would suffice to consider a Turing machine with 1 tape (respectively 2 stacks), since this type of Turing machine can simulate any other Turing machine (although not in real time). However it is known that various concrete problems (especially several pattern-matching problems) can be solved faster on a Turing machine that has more than one tape (see e.g. Hopcroft and Ullman, 1979, Maass, 1985, and Maass, Schnitger, Szemeredi, 1987). Therefore, and because it does not cause any extra work, we simulate rightaway an arbitrary Turing machine

M with *any* number k of stacks by an SNN $\mathcal{N}_M$.

At any computation step the Turing machine M may POP or PUSH a symbol on each of its k stacks. We assume for simplicity that the stack-alphabet of M is binary (i.e. M can push 0 or 1 on each stack, and pop a binary symbol, or receive the signal "bottom-of-stack" if the stack is empty.) Furthermore we assume that the input for the computation of M is given as the initial content of the first one of the k stacks, and that the output of M consists of the final content of the last one of the k stacks (at the moment when the machine halts).

If Q is the (finite) set of states of M, then after assigning a number in binary notation to each state in Q the transition function of M can be encoded by a function $F_M : \{0,1\}^{\lceil \log |Q| \rceil + k} \to \{0,1\}^{\lceil \log |Q| \rceil + k}$. We assume here that the state of M indicates on which of the stacks a POP or PUSH has to be carried out.

Thus in order to simulate the finite control of M by an SNN, it suffices to employ a module that can compute an arbitrary given function from $\{0,1\}^{\lceil \log |Q| \rceil + k}$ into itself. We assume here that the $\lceil \log |Q| \rceil + k$ input- and output-bits of this functions are stored in a corresponding number of oscillators with two states (dormant/oscillating). According to Lupanov, 1973, one can compute *any* function $F : \{0,1\}^{\lceil \log |Q| \rceil + k} \to \{0,1\}^{\lceil \log |Q| \rceil + k}$ on a feedforward threshold circuit with $O(|Q|^{1/2} \cdot 2^{k/2})$ gates. In addition, Horne and Hush, 1993 have shown that any such function F can be computed by a threshold circuit of depth 4 with $O(|Q|^{1/2} \cdot 2^{k/2} \cdot (\log |Q| + k))$ gates, using only weights and thresholds from $\{-1, 0, 1\}$. Hence our previously described simulation of an arbitrary threshold circuit on an SNN in subsection 2.5 allows us to simulate in $\mathcal{N}_M$ the finite control of M with a module of $O(|Q|^{1/2} \cdot 2^{k/2})$ neurons (provided the SNN may use arbitrarily large weights). Furthermore the quoted result by Horne and Hush in conjunction with our construction in subsection 2.5 implies that with $O(|Q|^{1/2} \cdot 2^{k/2} \cdot (\log |Q| + k))$ neurons one can implement in $\mathcal{N}_M$ the finite control of M in such a way that only very simple weights from $[0,1]$ are needed in $\mathcal{N}_M$, and that the simulation of each computation step of M requires only $O(1)$ "machine-cycles" of $\mathcal{N}_M$. More precisely, each computation step of M is simulated by $\mathcal{N}_M$ in a time interval in which the pace-maker PM fires $\leq K$ times, where K is some absolute constant that is independent of $|Q|, k$, the length of the current input of M, and the number of the previously simulated computation steps of M.

Apart from the finite control component, the SNN $\mathcal{N}_M$ consists of a module of $O(1)$ neurons for each of the k stacks, and $O(1)$ neurons which implement the pacemaker PM. In addition $\mathcal{N}_M$ uses $O(\log |Q| + k)$ neurons for other oscillators that serve as temporary registers for bits. Thus $\mathcal{N}_M$ consists altogether of at most $O(|Q|^{1/2} \cdot 2^{k/2} \cdot (\log |Q| + k))$ neurons, and the simulation of any computation step of M involves at most $O(|Q|^{1/2} \cdot 2^{k/2} \cdot (\log |Q| + k))$ firings of neurons in $\mathcal{N}_M$. After $\mathcal{N}_M$ has simulated every computation step of M on the current input $x(j) \in \{0,1\}^*$, it has generated on an oscillator O_S, which corresponds to the stack S on which

M writes its output $y(j) = \langle \tilde{b}_1, \ldots, \tilde{b}_{\tilde{\ell}} \rangle$, a phase-difference $\varphi_S = \sum_{i=1}^{\tilde{\ell}} \tilde{b}_i \cdot 2^{-i-c}$ with regard to the pacemaker PM. $\mathcal{N}_M$ outputs two spikes, where one is generated by PM and the other one by O_S, before receiving its next input. Since for fixed M the parameters $|Q|$ and k can be viewed as constants, $\mathcal{N}_M$ just uses $O(1)$ spikes for the simulation of each computation step of M. Hence $\mathcal{N}_M$ simulates M in real-time.

2.9 Weight-to-Phase Transformation

At this point the only missing link for the construction of the desired SNN $\mathcal{N}_{TM}(d)$ is a module which allows us to generate from suitable weights of an SNN the encoding of arbitrarily long (even infinitely long) bit strings, which may for example represent the program of a Turing machine, or an infinitely long "look-up table". The here constructed weight-to-phase transformation module will be able to generate within a fixed number of "machine cycles" any given phase-difference $\varphi = \sum_{i=1}^{\ell} b_i \cdot 2^{-i-c}$ of an oscillator (for arbitrary $\ell \in \mathbf{N} \cup \{\infty\}$ and $b_i \in \{0, 1\}$) from suitable weights between 0 and 1. Furthermore these weights can be chosen to be rational if $\ell \in \mathbf{N}$. This module will exploit effects of the firing mechanism of a neuron in an SNN that are closely related to those that we had used in subsection 2.6 in order to multiply the phase of an oscillator with a constant factor. In order to allow a *unique* decoding of *infinitely long* bit sequences from phase-differences φ we adapt the convention that $b_{2i} = 0$ for all $i \in \mathbf{N}$ in case that $\ell = \infty$.

We consider the same configuration with neurons u_1, u_2, v and an inhibition module as for MULTIPLY(β) in subsection 2.6. However instead of shifting the firing time of u_2, we are now interested in the consequences of multiplying the weight on the edge from u_1 to v with some factor $w \in [0, 1]$ (see Figure 9). We

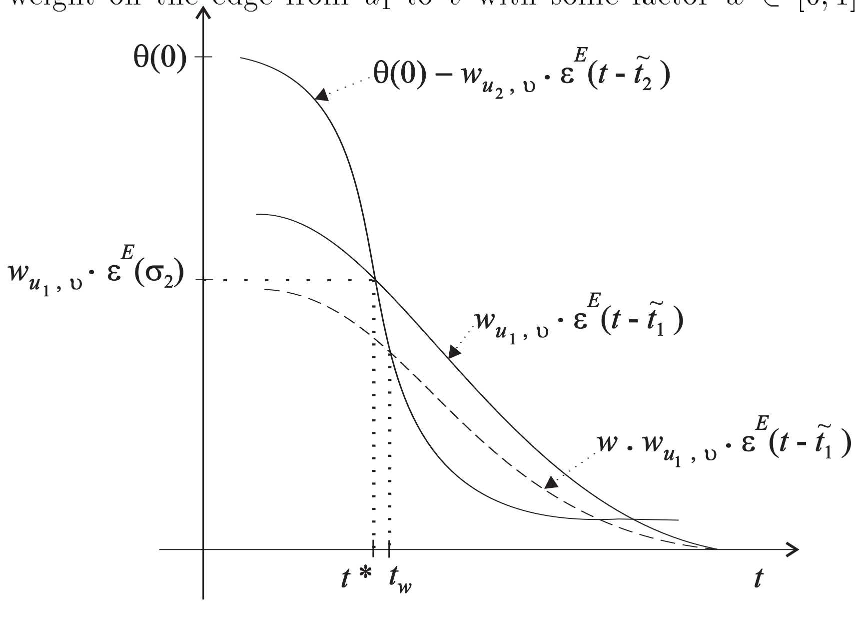

Figure 9: *Mechanism of the weight-to-phase transformation module.*

choose values for the delays $\Delta_{u_i,v}$ so that for $\tilde{t}_i := t_i + \Delta_{u_i,v}$ there exists some $t^* \geq \max(\tilde{t}_1, \tilde{t}_2)$ with $t^* - \tilde{t}_1 = \sigma_2$ and $t^* - \tilde{t}_2 = \sigma_1$. Furthermore we choose positive weights $w_{u_i,v}$ so that $w_{u_2,v} \cdot s_{up} = 2w_{u_1,v} \cdot s_{down}$ and

$$w_{u_1,v} \cdot \varepsilon^E(t^* - \tilde{t}_1) + w_{u_2,v} \cdot \varepsilon^E(t^* - \tilde{t}_2) \;\; = \;\; \Theta(0) \ .$$

In order to analyze the consequences of multiplying the weight $w_{u_1,v}$ with some $w \in [0,1]$, we consider for arbitrary $w \in [0,1]$ the point $t_w > t^*$ which satisfies

$$w \cdot w_{u_1,v} \cdot \varepsilon^E(t_w - \tilde{t}_1) + w_{u_2,v} \cdot \varepsilon^E(t_w - \tilde{t}_2) \;\; = \;\; \Theta(0) \ .$$

Together with the preceding equations and conditions (3) and (4) from our basic assumptions on ε^E this yields

$$(w-1) \cdot w_{u_1,v} \cdot \varepsilon^E(t^* - \tilde{t}_1) - w \cdot w_{u_1,v} \cdot s_{down} \cdot (t_w - t^*) + 2w_{u_1,v} \cdot s_{down} \cdot (t_w - t^*) = 0,$$

or equivalently

$$t_w - t^* \;\; = \;\; \frac{(1-w) \cdot w_{u_1,v} \cdot \varepsilon^E(\sigma_2)}{(2-w) \cdot w_{u_1,v} \cdot s_{down}} \;\; = \;\; \frac{(1-w) \cdot \varepsilon^E(\sigma_2)}{(2-w) \cdot s_{down}} \ .$$

Then analogous arguments as in subsection 2.6 show that if $w \in (0,1]$ is chosen so that the right hand side of this equation has a value in $[0, L/2]$, then the value for t_w which results from this equation is in fact the uniquely determined firing time of v in $[t^*, \ t^* + L/2]$ if the weight on the edge $\langle u_1, v \rangle$ is multiplied with w. In particular the value $t_w - t^* = L/2$ of the shift in the firing time of v is achieved for

$$w_L \, : \, = \;\; \frac{\varepsilon^E(\sigma_2) - L \cdot s_{down}}{\varepsilon^E(\sigma_2) - L \cdot s_{down}/2} \ .$$

Thus $w_L \in [0,1)$, and the function $w \mapsto t_w - t^*$ maps $[w_L, 1]$ one-one onto $[0, L/2]$.

The inverse of this map is defined by

$$t_w - t^* \;\; \mapsto w \;\; := \;\; \frac{\varepsilon^E(\sigma_2) - 2(t_w - t^*) \cdot s_{down}}{\varepsilon^E(\sigma_2) - (t_w - t^*) \cdot s_{down}} \ .$$

One can derive from the basic assumptions on Θ and ε^E that $w_{u_1,v} \in \mathbf{Q}$. Hence the preceding formula in combination with these basic assumptions implies that one can achieve *any rational* phase-shift $t_w - t^* \in [0, L/2]$ with a *rational* weight $w \cdot w_{u_1,v}$ on the edge $\langle u_1, v \rangle$.

Finally, by our choice of c one has $\sum\limits_{i=1}^{\ell} b_i \cdot 2^{-i-c} \in [0, L/2]$ for *any* values of $\ell \in \mathbf{N} \cup \{\infty\}$ and $b_i \in \{0,1\}$. Hence in a preprocessing phase of an SNN any given finite or infinite bit-sequence $\langle b_1, b_2, \ldots \rangle$ can be "loaded" (with only $O(1)$ spikes involved) from the value of a certain weight of the SNN into the form of a phase-difference $\varphi_S = \sum\limits_{i=1}^{\ell} b_i \cdot 2^{-i-c}$ of an oscillator O_S. For that purpose one has to arrange that the considered firings of neurons u_1 and u_2 (as well as of the involved inhibition module, see the corresponding construction in subsection 2.6) are triggered by EPSP's from the pacemaker PM. Thus we have shown that the weights of an SNN can essentially play the role of a "read-only memory" of unlimited capacity.

2.10 Construction of $\mathcal{N}_{TM}(d)$

In this last part of the proof of Theorem 2.1 we construct an SNN $\mathcal{N}_{TM}(d)$ which has those properties that are claimed in Theorem 2.1. Let $d \in \mathbf{N}$ be any given constant. Let $M_{\mathcal{U}}$ be a "universal Turing machine" with $d+1$ tapes that can simulate any Turing machine with d tapes in real-time. More precisely, $M_{\mathcal{U}}$ is a Turing machine which receives two finite binary strings x and e on two different tapes as input, and which simulates for any $e \in \{0,1\}^*$ the d-tape Turing machine whose program is encoded by e in real-time on input x (with some suitable default-convention for the case that e is not the encoding of some Turing-machine program). The construction of such universal Turing machines $M_{\mathcal{U}}$ is a standard part of the proof of the time-hierarchy theorem for Turing machines (see e.g. Hopcroft and Ullman, 1979, or Lewis and Papadimitriou, 1981). The desired SNN $\mathcal{N}_{TM}(d)$ will basically be that SNN which one gets by applying the construction from subsection 2.8 to the Turing machine $M := M_{\mathcal{U}}$, but with $2d+2$ stacks instead of the $d+1$ tapes.

The only additional work that remains to be done in order to satisfy the claim of Theorem 2.1 is to change the way in which $\mathcal{N}_{M_{\mathcal{U}}}$ receives its input. Ordinarily $\mathcal{N}_{M_{\mathcal{U}}}$ would expect to get its second input $e = \langle e_1, \ldots, e_\ell \rangle \in \{0,1\}^*$ in the same way as its first input $x \in \{0,1\}^*$, in the form of two input spikes with time-distance $\sum\limits_{i=1}^{\ell} e_i \cdot 2^{-i-c}$.

In contrast to that, the constructed SNN $\mathcal{N}_{TM}(d)$ only receives a single input x in the form of a time-difference between two input spikes. On the other hand its weights may depend on the simulated Turing machine M. Thus we may choose a rational weight $w \in [0,1]$ that can be transformed with the help of the module from subsection 2.9 into a phase-difference $t_w - t^* = \sum\limits_{i=1}^{\ell} e_i \cdot 2^{-i-c}$. This transformation can be carried out in a preprocessing phase within $O(1)$ firings of PM. After that, the computation of $\mathcal{N}_{TM}(d)$ proceeds exactly like that of $\mathcal{N}_{M_{\mathcal{U}}}$.

In order to prove the second part of the claim of Theorem 2.1, one exploits the obvious fact that *any* function $F : \{0,1\}^* \to \{0,1\}^*$ can be computed by a Turing machine M_F with infinitely many bits of "advice", i.e. by a Turing machine M_F which has at the beginning of each computation on one of its tapes the same infinite sequence $\langle e_i \rangle_{i \in \mathbf{N}}$ of bits $e_i \in \{0,1\}$ as initial tape inscription. This sequence $\langle e_i \rangle_{i \in \mathbf{N}}$ may for example encode a look-up-table for all pairs $\langle x, F(x) \rangle$, $x \in \{0,1\}^*$. We may assume that $\langle e_i \rangle_{i \in \mathbf{N}}$ also encodes the program of the Turing machine M_F, and that M_F altogether has only 2 tapes. As usual, the Turing machine M_F receives on another tape the input $x \in \{0,1\}^*$. In order to simulate this Turing machine M_F on the SNN $\mathcal{N}_{TM}(d)$, we just have to equip $\mathcal{N}_{TM}(d)$ with a suitable real weight $w \in [0,1]$, that can be transformed (as described in section 2.9) in a preprocessing phase within $O(1)$ firings of PM into the phase-difference $t_w - t^* = \sum_{i=1}^{\infty} e_i \cdot 2^{-2i-c}$ of an oscillator. After that, $\mathcal{N}_{TM}(d)$ will simulate the computation of the Turing machine M_F (with initial tape content $\langle e_i \rangle_{i\in\mathbf{N}}$ on one of its tapes) in the usual manner. Thus $\mathcal{N}_{TM}(d)$ will output $F(x)$ for any given input $x \in \{0,1\}^*$. Hence $\mathcal{N}_{TM}(d)$ can compute the (arbitrarily given) function $F : \{0,1\}^* \to \{0,1\}^*$. This concludes the proof of Theorem 2.1. ∎

An important measure for the complexity of a neural network in the context of *learning* is its Vapnik-Chervonenkis dimension (VC-dimension), see Vapnik and Chervonenkis, 1971. Various results from statistical theory suggest that the VC-dimension of a neural net is proportional to the number of examples needed to *train* that neural net (for references and a brief survey see e.g. Maass, 1995a).

Corollary 2.2 *One can construct with any type of neurons whose response- and threshold-functions satisfy our basic assumptions an SNN $\mathcal{N}$ of finite size, so that the VC-dimension of the class of boolean functions that are computable on $\mathcal{N}$ (with different assignments of rational values from $[0,1]$ to its weights) is infinite.*

A proof of the following result is contained as a special case in the proof of Theorem 2.1 (see especially subsection 2.8).

Corollary 2.3 *Any deterministic finite automaton with q states can be simulated in real-time (both for decision problems, or with intermediate output as a Mealy- or Moore-machine) by an SNN with $O(q^{1/2})$ neurons (respectively with $O(q^{1/2} \cdot \log q)$ neurons if only weights from $[0,1]$ are permitted).*

The following corollary exhibits another result of independent interest that was shown in the preceding proof (subsection 2.5).

Corollary 2.4 *One can construct with any type of neurons whose response- and threshold-functions satisfy our basic assumption for any given feedforward boolean threshold circuit C with arbitrary weights, s gates and d hidden layers an SNN S_C with $O(s)$ neurons that simulates any computation of C within a time interval of length $O(d)$. Furthermore one can also simulate C within time $O(d)$ by an SNN S'_C with polynomial(s) neurons that only uses weights $w \in [0,1]$.*

Finally we observe that an application of the techniques from the proof of Theorem 2.1 to SNN's with discrete time (see the definitions in section 1) yields the following result.

Corollary 2.5 *One can construct for any Turing machine M with any type* *of neurons whose response- and threshold functions satisfy our basic assumptions an SNN $\mathcal{N}_M$ so that for any $s \in \boldsymbol{N}$ the SNN $\mathcal{N}_M$ with discrete firing times from $\{i \cdot \mu : i \in \boldsymbol{N}\}$ for some μ with $1/\mu = 2^{s+O(1)}$ and $\Delta_{\max} - \Delta_{\min} \geq 2\mu$ can simulate in real-time arbitrary computations of M that involve at most s tape cells of M.*

For the **proof** of Corollary 2.5 one exploits that because of the condition $\Delta_{\max} - \Delta_{\min} \geq 2\mu$ the same construction as in subsection 2.2 yields modules that achieve any given real-valued (!) delay $\geq \Delta_{\min}$. With the help of such delay modules one can then arrange that the time points t^* in the subsequent constructions of other modules, as well as the time points when the EPSP's reach their maximal value $\varepsilon_{\max}$ (for the simulation of a threshold circuit), all belong to the set $\{i \cdot \mu \ : \ i \in \mathbf{N}\}$. For the simulation of a stack of a Turing machine M the construction from the proof of Theorem 2.1 works without changes for SNN's with discrete time steps of length μ, provided that $2^{-\ell-c} \in \{i \cdot \mu : i \in \mathbf{N}\}$ for the maximal length ℓ of any bit-string that is stored in a stack of M. ∎

3 Beyond Turing Machines

We have shown in Theorem 2.1 that one can build from arbitrary neurons, whose response- and threshold-functions satisfy certain basic assumptions, an SNN that can simulate any Turing machine. However SNN's are strictly more powerful than Turing machines for two reasons:

i) An SNN can receive *real* numbers as input, and give *real* numbers as output (in the form of time-differences between pairs of spikes).

ii) We had constructed in subsection 2.6 modules for an SNN that can carry out the operations COMPARE ($\geq \alpha$) and MULTIPLY(β), for a wide range of constants α and β, applied to arbitrary real-valued arguments φ from a certain interval. If one applies for example such operation to a phase of the form $\varphi = \sum_{i=1}^{\ell} b_i \cdot 2^{-i-c}$, then such module executes with $O(1)$ spikes an operation that involves the *whole* bit-string $\langle b_1, \ldots \rangle$ of arbitrary length $\ell \in \mathbf{N} \cup \{\infty\}$. In contrast to that, any Turing machine operation can affect at best a constant number of its stored bits.

In this section we will show that in addition one can construct modules for an SNN that ADD, SUBTRACT, or COMPARE any two real valued phase-differences $\varphi_1, \varphi_2 \in [0, L/4]$ of two different oscillators. This result turns out to be quite important, since in combination with i) and ii) it implies that one can simulate in real-time on an SNN any RAM with finitely many registers that stores in its registers arbitrary real numbers of bounded absolute value, and which uses arbitrary instructions of the form COMPARE, MULTIPLY(β), ADD, SUBTRACT. Furthermore such SNN can be built with any type of neurons whose response- and threshold-functions satisfy the basic assumptions from the beginning of section 2.

On the other hand according to Maass, 1994b, 1995c, any SNN with *arbitrary* piecewise linear response- and threshold-functions can be simulated in real-time by the same type of RAM. Hence, the computational power of these RAM's (which we will call N-RAM's because of their close relationships to neural networks) matches exactly that of SNN's whose response- and threshold-functions are piecewise linear *and* satisfy our basic assumptions.

One can also show through mutual real-time simulations (see Maass, 1995b, 1995c) that the computational power of N-RAM's (and hence of the abovementioned SNN's) matches exactly that of recurrent *analog* neural nets with discrete time and piecewise linear activation functions (see Siegelmann and Sontag, 1992). More precisely: any analog neural net with any piecewise linear activation functions can be simulated in real-time by an N-RAM; for the simulation of N-RAM's by analog neural nets one can employ for example the linear saturated activation function together with the heaviside activation function in the analog neural net. This result implies as a side-result that these two activation functions together are "universal" for all piecewise linear activation functions in recurrent analog neural nets (since they allow such net to simulate in real-time any other recurrent analog neural net with arbitrary piecewise linear activation functions). Hence N-RAM's provide also a very useful *intermediate link* for the comparison of SNN's (modeling *spike coding*) and analog neural nets (modeling *frequency coding*).

We defer the detailed discussion of N-RAM's (which are somewhat weaker than the well-known model of Blum, Shub, Smale, 1989, and also related to the computational model considered in Koiran, 1993) and the proofs of the abovementioned results to a subsequent article (Maass, 1995c). However we will describe in this section the construction of SNN-modules for the operations ADD, SUBTRACT, and COMPARE, since those constructions are closely related to the preceding constructions in this article. These constructions provide the tools for the real-time simulation of N-RAM's by SNN's.

Consider two oscillators O_1 and O_2 of an SNN, both with oscillation-period π_{PM}. Let φ_i be the phase-difference between O_i and the pacemaker PM, $i = 1, 2$. We construct a module that receives a spike from each of the oscillators O_1 and O_2, and which is then able to kick-off a third oscillator O with oscillation period π_{PM} in such a way that it will have phase-difference $\varphi_1 + \varphi_2$ to PM. This module for the operation ADD employs a similar arrangement of three neurons u_1, u_2 and v as the modules for COMPARE($\geq \alpha$) and MULTIPLY(β) that were constructed in subsection 2.6. We assume that neuron u_i is triggered by a spike from oscillator O_i to fire at a certain time t_i. We choose delays $\Delta_{u_i,v}$ in such a way that for $\tilde{t}_i := t_i + \Delta_{u_i,v}$ there exists some $t^* \geq \max(\tilde{t}_1, \tilde{t}_2)$ so that $t^* - \tilde{t}_1 = t^* - \tilde{t}_2 = \sigma_1$ in case that $\varphi_1 = \varphi_2 = 0$. We choose $w > 0$ so that $2w \cdot \varepsilon^E(\sigma_1) = \Theta(0)$, and we set $w_{u_1,v} = w_{u_2,v} = w$. We also add an inhibition-module, which makes it impossible for v to fire within the time-interval

$[t^* - \tau_{end},\ \ t^* - L/2)$ for any values of φ_1, φ_2, and which has no influence on $P_v(t)$ for $t \geq t^*$ (as in the construction for MULTIPLY(β) in subsection 2.6).

Then for arbitrary values $\varphi_1, \varphi_2 \in [0, L/2]$ the neuron v fires at a time $t_\Sigma \in [0, L/2]$ such that

$$w \cdot s_{up} \cdot (t_\Sigma - (t^* + \varphi_1)) + w \cdot s_{up} \cdot (t_\Sigma - (t^* + \varphi_2)) \quad = \quad 0 \ ,$$

or equivalently

$$t_\Sigma - t^* \quad = \quad \frac{\varphi_1 + \varphi_2}{2}$$

(see Figure 10). The factor 1/2 of $\varphi_1 + \varphi_2$ can be removed with the help of a

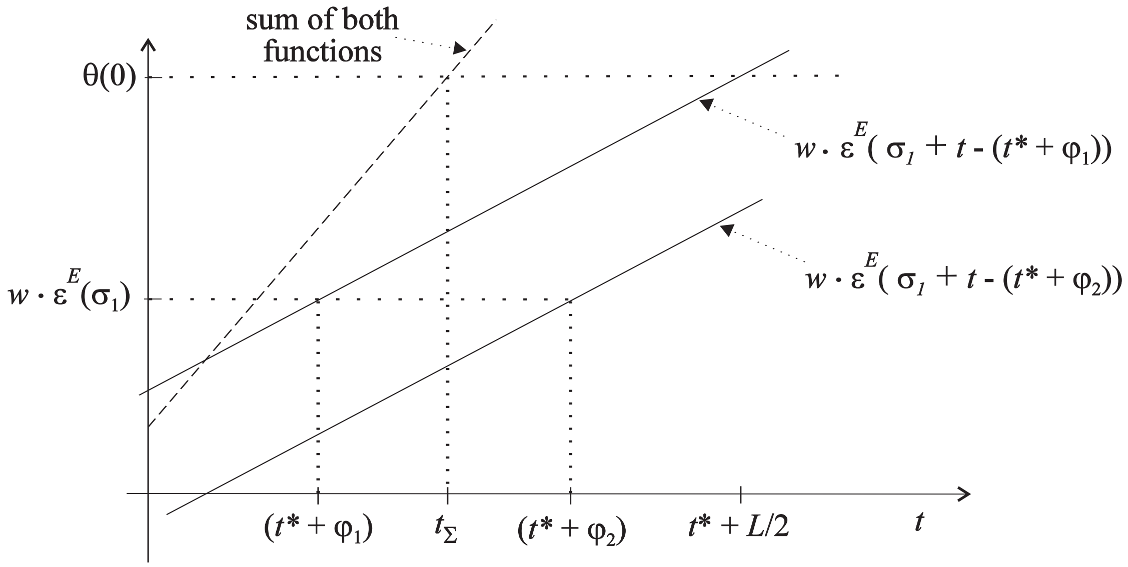

Figure 10: *Mechanism of the module for ADD.*

subsequent module for MULTIPLY(2) (see subsection 2.6). In this way the here constructed module for ADD can generate an output-spike at time $i \cdot \pi_{\mathrm{PM}} + (\varphi_1 + \varphi_2)$ for some $i \in \mathbf{N}$.

The construction of a module that computes the *difference* $\varphi_1 - \varphi_2$ of the phase-differences φ_1, φ_2 of two modules O_1 and O_2 with $\varphi_1 \geq \varphi_2$ is quite similar. For arbitrary given values $\varphi_1, \varphi_2 \in [0, L/2]$ with $\varphi_1 \geq \varphi_2$ we first employ a module MULTIPLY(1/2) which replaces φ_1 by $\tilde{\varphi}_1 := \varphi_1/2$. For an arrangement of neurons u_1, u_2, v similarly as for ADD we choose delays $\Delta_{u_i,v}$ so that $t^* - \tilde{t}_1 = \sigma_1$ and $t^* - \tilde{t}_2 = \sigma_2$ in case that $\varphi_1 = \varphi_2 = 0$, and weights $w_{u_1,v},\ \ w_{u_2,v}$ so that $w_{u_1,v} \cdot s_{up} = 2 w_{u_2,v} \cdot s_{down}$ and $w_{u_1,v} \cdot \varepsilon^E(\sigma_1) + w_{u_2,v} \cdot \varepsilon^E(\sigma_2) = \Theta(0)$. Furthermore we employ an inhibition module which makes it for any values of $\varphi_1, \varphi_2 \in [0, L/2]$ impossible for neuron v to fire within the time-interval $[t^* - \tau_{end}, \quad t^* - L/2]$, but which has no influence on $P_v(t)$ for $t \geq t^*$.

Then for any phase-differences $\varphi_1, \varphi_2 \in [0, L/2]$ with $\varphi_1 \geq \varphi_2$ the phase-difference φ_1 is first transformed to $\tilde{\varphi}_1 = \varphi_1/2$. Neuron u_1 receives a spike with phase-difference $\tilde{\varphi}_1$, and u_2 receives a spike with phase difference φ_2. The resulting firing time t_Δ of neuron v is determined by (see Figure 11)

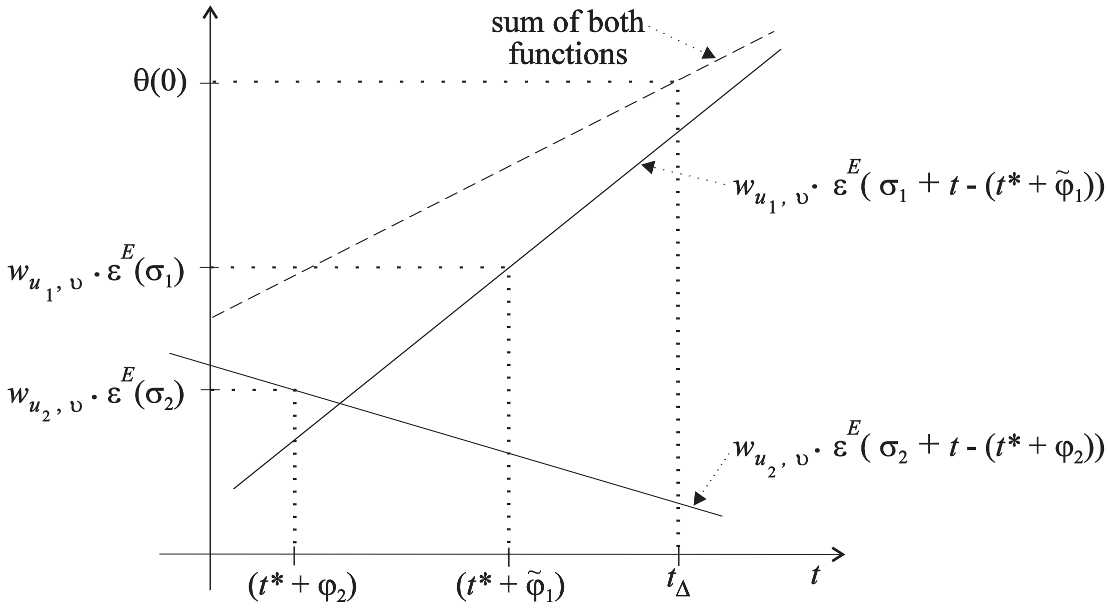

Figure 11: *Mechanism of the module for SUBTRACT.*

$$w_{u_1,v} \cdot s_{up} \cdot (t_\Delta - (t^* + \tilde{\varphi}_1)) - w_{u_2,v} \cdot s_{down} \cdot (t_\Delta - (t^* + \varphi_2)) \quad = \quad 0 \quad .$$

This yields

$$t_\Delta - t^* \quad = \quad 2\tilde{\varphi}_1 - \varphi_2 \quad = \quad \varphi_1 - \varphi_2 \quad .$$

Finally, it is easy to see that the module for COMPARE($\geq \alpha$) from subsection 2.6 in combination with the preceding module for SUBTRACT allows us to build a module for the test COMPARE, i.e. a module which decides for any two given phase-differences $\varphi_1, \varphi_2 \in [0, L/4]$ of two oscillators O_1 and O_2 with oscillation-period π_{PM} whether $\varphi_1 \geq \varphi_2$. For that purpose one first transforms φ_1 with the help of a delay module to $\varphi_1' := \varphi_1 + L/4$. It is then clear that $\varphi_1' \geq \varphi_2$, and the module for SUBTRACT can be employed to compute $\varphi_1' - \varphi_2 = \varphi_1 - \varphi_2 + L/4$. With the help of a subsequent module for COMPARE($\geq L/4$) we can then decide whether $\varphi_1 - \varphi_2 + L/4 \geq L/4$, i.e. whether $\varphi_1 \geq \varphi_2$.

Of course one can also build directly a module for COMPARE by using a variation of the construction for COMPARE($\geq \alpha$) in subsection 2.6.

4 Variations of the Constructions for Related Models of Spiking Neurons

We have assumed for the constructions in the preceding two sections that the response- and threshold-functions are stereotyped, i.e. that apart from their individual delays $\Delta_{u,v}$ the functions $\varepsilon_{u,v}$ and Θ_v all have the sampe shape. This assumption is convenient, but not really necessary for the preceding constructions. The same constructions can also be carried out if these functions are *different* for different edges $\langle u, v\rangle \in E$ and different $v \in V$. More precisely, it suffices to assume that the response-functions $\varepsilon_{u,v}$ are defined with the help of individual delays $\Delta_{u,v}$ *and individual* functions $\varepsilon^E_{u,v}$ respectively $\varepsilon^I_{u,v}$, so that $\varepsilon_{u,v}(x) = 0$ for $x \in [0, \Delta_{u,v}]$ and $\varepsilon_{u,v}(\Delta_{u,v} + x) = \varepsilon^E_{u,v}(x)$, respectively $\varepsilon_{u,v}(\Delta_{u,v} + x) = \varepsilon^I_{u,v}(x)$ in the case of an IPSP, where the functions $\varepsilon^E_{u,v}, \varepsilon^I_{u,v}, \Theta_v$ satisfy the basic assumptions from the beginning of section 2. However these functions $\varepsilon^E_{u,v}, \varepsilon^I_{u,v}$, and Θ_v may be arbitrarily *different*, with different values of the parameters $\tau_{ref}, \tau_{end}, \sigma_j, \tau_j, L, s_{up}, s_{down}$, for different neurons u, v (in fact one may assume that these functions are chosen by an "adversary"). Under these relaxed conditions we have to assume however that *we* can choose arbitrarily large delays $\Delta_{u,v}$ and weights $w_{u,v}$ *after* the individual functions $\varepsilon^E_{u,v}, \varepsilon^I_{u,v}$ and Θ_v are given to us. Of course one can trade-off parts of the latter condition against some quite reasonable conditions on the individual functions $\varepsilon^E_{u,v}, \varepsilon^I_{u,v}$, and Θ_v.

One can also replace the basic assumptions at the beginning of section 2 by some alternative assumptions about $\varepsilon^E_{u,v}, \varepsilon^I_{u,v}$, and Θ_v. For example one can postulate the existence of suitable linear segments of $\varepsilon^I_{u,v}$ or Θ_v, and then exploit at the neuron v in the module-constructions of sections 2 and 3 a "timing-race" between an EPSP and an IPSP, or between an EPSP and the declining part of Θ_v (instead of the race between two EPSP's). Without a "reset" at each firing of neuron v (see below) one needs however for the latter option (EPSP's versus Θ_v) more specific assumptions about these functions, in order to control undesired side-effects that may result from the end-segments of EPSP's that caused the *preceding* firing of v.

We also would like to point out that the full power of the module COMPARE($\geq \alpha$) from subsection 2.6 is actually not needed if one just wants to simulate Turing machines on an SNN. If one employs a less concise encoding of bit-strings by assuming that also $b_{2i} = 0$ for all $i \leq \ell/2$ for all *finite* bit-strings $\langle b_1, \ldots, b_\ell\rangle$ that are encoded in the phase-difference $\varphi = \sum\limits_{i=1}^{\ell} b_i \cdot 2^{-i-c}$ of an oscillator, it is guaranteed that $\varphi \geq 2^{-1-c}$ or $\varphi \leq 2^{-2-c}$ (independently of ℓ and of the values of the $b_i \in \{0, 1\}$). This "gap" of fixed length between the possible values of φ allows us to carry out the test whether $b_1 = 1$ just with the help of delay- and inhibition-modules (instead of using the more subtle mechanism of COMPARE($\geq \alpha$)). But the module for COMPARE($\geq \alpha$) is of independent interest, since it shows in the context of section 3 that also *discontinuous* real-valued functions can be computed on an SNN.

The implicit assumptions about the firing mechanism of neurons in the version of the SNN-model from section 1 ignore the well-known "reset" and "adaptation" phenomena of neurons. However one can easily adjust the definition of the SNN-model so that it also takes these features into account. In order to model a *reset* of a neuron at its moment of firing, one can adjust the definition of the set F_v of firing times of a neuron v by deleting (or modifying) in the definition of $P_v(t)$ those EPSP's and IPSP's from presynaptic neurons u that had already arrived at v *before* the most recent firing of v.

Adaptation of a neuron v refers to the observation that the firing-rate of a biological neuron may decline after a while even if the incoming excitation (i.e. $P_v(t)$) remains at a constant high level), see for example Kandel et al., 1991. This effect can be reflected in the SNN-model by replacing the term $\Theta_v(t-s)$ in the definition of the set F_v of firing times by a sum over $\Theta_v(t-s)$ for *several* recent firing times $s \in F_v$ (and by assuming that $\Theta_v(x)$ returns only relatively slowly to its initial value $\Theta(0)$).

We would like to point out that all of our constructions in sections 2 and 3 are compatible with our abovementioned changes in the SNN-model for modelling the *reset* and *adaptation* of neurons. The reason for this is that we can arrange in the constructions of sections 2 and 3 that all "relevant" firings of a neuron v are spaced so far apart that reset and adaption of v have no effect on those *critical* firing times.

Regarding the simulation of *threshold circuits* by SNN's (see subsection 2.5) we would like to point out that the corresponding SNN-module can be constructed with *fewer* neurons if one makes further assumptions about the shape of EPSP- and IPSP-response-functions. For example one can simulate directly a threshold gate $T^{\underline{\alpha}}$ with weights α_i of *different* sign in a similar way as we have simulated *monotone* threshold gates $T^{\underline{\alpha}}$ in subsection 2.5, provided that the EPSP's (modelling inputs with positive weights) and IPSP's (modelling inputs with negative weights) move linearly within the same time-span from 0 to their extremal values.

Finally, we would like to point out that the class of piecewise *constant* functions (i.e. the class of step-functions) provides an example for a class of response- and threshold-functions which do *not* satisfy our basic assumptions from section 2, but which can still be used to build for any Turing machine M an SNN $\mathcal{N}_{M'}$ that can simulate M (although not in real-time). We assume here that the response-functions are piecewise constant (but not identically zero), and that the threshold-functions are arbitrary functions (e.g. piecewise constant) that satisfy condition (1) of our basic assumptions. One can then build oscillators, as well as delay-, inhibition-, and synchronization-modules, in the same way as in section 2, and one can also simulate arbitrary threshold circuits in the same way. Furthermore one can use the phase-difference between an oscillator O with the same oscillation period π_{PM} as the pacemaker PM in order to simulate a *counter*. For that purpose one employs a delay module D with a suitable delay $\rho > 0$ (so that $k \cdot \rho = \ell \cdot \pi_{\mathrm{PM}}$ for any $k, \ell \in \mathbf{N}$ implies that $k = \ell = 0$). One can then use the phase-difference between O and PM to record how often the "spike in O" has been directed in the course of the computation through this delay module D. Hence one can store in the SNN an arbitrary natural number k, which can be incremented and decremented by suitable modules. In order to decide whether $k = 0$, one needs a module that can carry out a special case of the operation COMPARE. Such module cannot be built in the same way as in sections 2 respectively 3, but one can employ directly the "jump" in the here considered piecewise constant response- functions in order to test whether two neurons fire exactly at the same time.

It is well-known (see Hopcroft and Ullman, 1979) that any Turing machine M can be simulated (although not in real-time) by a machine M' that has no tapes or stacks, but two counters. The preceding argument shows that such M' (in fact: a machine with any finite number of counters) can be simulated in real-time by some finite SNN $\mathcal{N}_{M'}$ with *piecewise constant* response- and threshold-functions.

The effect of the shape of postsynaptic potentials on the computational power of networks of spiking neurons is investigated more thoroughly in Maass and Ruf, 1995. It is shown there, that computations with single spikes in networks of spiking neurons become substantially slower if they cannot make use of *increasing* and *decreasing* linear segments of EPSP's.

5 Conclusion

We have analyzed the computational power of a simple formal model SNN for networks of spiking neurons. In particular we have shown that if the response- and threshold-functions of the SNN satisfy some rather weak basic assumptions (see section 2), then one can build modules that can *synchronize* the spiking of different network parts, as well as modules that can *multiply* the phase difference between two oscillators with any given constant, and *add*, *subtract* or *compare* the phase differences of different oscillators (see the constructions in sections 2 and 3). With the help of these quite powerful computational operations an SNN can simulate in real-time for boolean-valued input any Turing machine, and for real-valued input any N-RAM (a slightly weaker version of the model of Blum, Shub, Smale, 1989; see section 3 of this article). On the side we would like to mention that these results also yield lower bounds for the VC-dimension of networks of spiking neurons, hence for the number of training examples needed for learning by such networks (see Maass, 1994b, 1995c). One immediate consequence of this type is indicated in Corollary 2.2 of this article.

The version of the model SNN with unlimited timing-precision (i.e. $T = \mathbf{R}^+$ in the definition in section 1) is insofar not biologically realistic, as it does not take the effects of noise into account. From that point of view our alternative version of this model with *discrete* firing times from $\{i \cdot \mu : i \in \mathbf{N}\}$ for some $\mu > 0$ is preferrable (since it allows us to represent an imprecise firing anywhere in the time-interval $(i \cdot \mu - \varepsilon, i \cdot \mu + \varepsilon)$ in the biological system by a "symbolic" firing at time $i \cdot \mu$). Therefore it is important to note that our results about SNN's with unlimited timing precision *induce* corresponding results for the computational power of SNN's with *discrete* firing times, as we have indicated in Corollary 2.5 (see Theorem 5 in Maass, 1994b, as well as Maass, 1995c, for further consequences of our results for SNN's with limited timing precision). In addition our constructions of SNN-modules for the operations ADD, SUBTRACT, and MULTIPLY(β) on time-differences between spikes appear to be quite robust, in the sense that they provide approximate implementations of these operations on time differences between spikes in various models for *real-valued* computations in networks of spiking neurons *with noise.*

The results of this article have two interesting consequences. One is, that in order to show that a network of spiking neurons can carry out some specific task (e.g. in pattern recognition or pattern segmentation, or solving some binding problem; see e.g. Malsburg and Schneider, 1986, or Gerstner et al., 1993) it now suffices to show that a threshold circuit, a finite automaton, a Turing machine or an N-RAM (see section 3) can carry out that task in an efficient manner. Furthermore the simulation results of this article allow us to relate the computational *resources* that are needed on the latter more convenient models (e.g. the required work space on a Turing machine) to the required resources needed by the SNN (e.g. the timing precision of the SNN, see Corollary 2.5). In other words, one may view N-RAM's and the other mentioned common computational models as "higher programming languages" for the construction of networks of spiking neurons. The real-time simulation methods of this article exhibit automatic methods for translating any program that is written in such higher programming language into the construction of a corresponding SNN. In this way the "user" of an SNN may choose to ignore all worrisome implementation details on SNN's such as timing (potentially at the cost of some efficiency). Furthermore the matching upper bound result for N-RAM's (see Maass 1995b, 1995c) shows that the corresponding "higher programming language" is able to exploit *all* computational abilities of SNN's.

Secondly, in combination with the corresponding *upper* bound results for SNN's with quite arbitrary response- and threshold-functions (and time-dependent weights) in Maass, 1995b, 1995c, the *lower* bounds of this article provide for a large class of response- and threshold-functions *exact* characterizations (up to real-time simulations) of the computational power of SNN's with real valued inputs, and for SNN's with bounded timing precision. As a consequence of these results, one can then also relate the computational power of SNN's to that of recurrent *analog* neural nets with various activation functions (see section 3), thereby throwing some light on the relationships between the computational power of models of neurons with *spike-coding* (SNN's) and models of neurons with *frequency-coding* (analog neural nets). Furthermore the combination of these lower and upper bound results shows that extremely simple response- and threshold-functions (such as for example those in Figure 2 in section 2) are *universal* in the sense that with these functions an SNN can simulate in real-time any SNN that employs *arbitrary* piecewise linear response- and threshold-functions. Equivalence-results of this type induce some structure in the "zoo" of response- and threshold-functions that are mathematically interesting or occur in biological neural systems, and they allow us to focus on those aspects of these functions which are *essential* for the computational power of spiking neurons.

Finally we would like to point out that since we have based all of our investigations on the rather fine notion of a *real-time simulation* (see section 1), our results provide information not just about the relationships between the *computational* power of the previously mentioned models for neural networks, but also about their capability to execute *learning* algorithms (i.e. about their *adaptive* qualities).

Acknowledgement

I would like to thank Wulfram Gerstner, John G. Taylor, and three anonymous referees for helpful comments.

References

M. Abeles. (1991) Corticonics: Neural Circuits of the Cerebral Cortex. *Cambridge University Press.*

A. Aertsen. ed. (1993) Brain Theory: Spatio-Temporal Aspects of Brain Function. *Elsevier.*

L. Blum, M. Shub, S. Smale. (1989) On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bull. Amer. Math. Soc.* 21(1): 1-46.

J. Buhmann, K. Schulten. (1986) Associative recognition and storage in a model network of physiological neurons. *Biol. Cybern.* 54: 319-335.

P. S. Churchland, T. J. Sejnowski. (1992) The Computational Brain. *MIT-Press.*

W. Gerstner. (1991) Associative memory in a network of "biological" neurons. *Advances in Neural Information Processing Systems, vol. 3, Morgan Kaufmann*: 84-90.

W. Gerstner, R. Ritz, J. L. van Hemmen. (1993) A biologically motivated and analytically soluble model of collective oscillations in the cortex. *Biol. Cybern.* 68: 363-374.

W. Gerstner, J. L. van Hemmen. (1994) How to describe neuronal activity:

spikes, rates, or assemblies? appears in: *Advances in Neural Information Processing Systems, vol. 6, Morgan Kaufmann*: 463-470.

W. Gerstner. (1995) Time structure of the activity in neural network models. *Physical Review E*, 51: 738-758.

A. Hajnal, W. Maass, P. Pudlak, M. Szegedy, G. Turan. (1993) Threshold circuits of bounded depth. *J. Comput. System Sci.* 46: 129-154.

J. E. Hopcroft, J. D. Ullman. (1979) Introduction to Automata Theory, Languages, and Computation. *Addison-Wesley.*

B. G. Horne, D. R. Hush. (1993) On the node complexity of neural networks. *Technical Report EECE 93-003, Dept. EECE, Univ. of New Mexico.*

K. T. Judd, K. Aihara. (1993) Pulse propagation networks: A neural network model that uses temporal coding by action potentials. *Neural Networks* 6: 203-215.

E. R. Kandel, J. H. Schwartz, T. M. Jessel. (1991) Principles of Neural Science. *Prentice-Hall.*

P. Koiran. (1993) A weak version of the Blum, Shub, Smale model. *Proc. of the 34th Annual IEEE Symp. on Found. of Comp. Sci., IEEE Computer Society Press*: 486-495.

H. R. Lewis, C. H. Papadimitriou. (1981) Elements of the Theory of Computation. *Prentice-Hall.*

W. Maass. (1985) Combinatorial lower bound arguments for deterministic and nondeterministic Turing machines. *Trans. Amer. Math. Soc.* 292: 675-693.

W. Maass. (1993) Bounds for the computational power and learning complexity of analog neural nets. *Proc. 25th Annual ACM Symposium on the Theory of Computing*: 335-344.

W. Maass. (1994a) Neural Nets with Superlinear VC-Dimension. *Proc. of The European Conference on Artificial Neural Networks 1994 (ICANN '94)*; journal version appeared in *Neural Computation* 6: 875-882.

W. Maass. (1994b) On the computational complexity of networks of spiking neurons (extended abstract). *TR 393 from May 1994 of the Institutes for Information Processing Graz*; to appear in the *Proceedings of the 1994 Conference on Neural Information Processing Systems (NIPS '94), MIT-Press.*

W. Maass. (1995a) Vapnik-Chervonenkis Dimension of Neural Nets. To appear in the *Handbook of Brain Theory and Neural Networks, M. A. Arbib, ed., MIT-Press.*

W. Maass. (1995b) Analog Computations on Networks of Spiking Neurons (Extended Abstract). *Submitted for publication.*

W. Maass. (1995c) Upper bounds for the computational power of networks of spiking neurons. *In preparation.*

W. Maass, B. Ruf. (1995) Consequences of the shape of postsynaptic potentials for the computational power of networks of spiking neurons. *Submitted for publication.*

W. Maass, G. Schnitger, E. Szemeredi. (1987) Two tapes are better than one for off-line Turing machines. *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*: 94-100.

A. Murray, L. Torassenko. (1994) Analogue Neural VLSI: A Pulse Stream Approach. *Chapman & Hall.*

H. T. Siegelmann, E. D. Sontag. (1992) On the computational power of neural nets. *Proc. 5th ACM-Workshop on Computational Learning Theory*: 440-449.

H. C. Tuckwell. (1988) Introduction to Theoretical Neurobiology, volumes 1 and 2. *Cambridge University Press.*

L. G. Valiant. (1994) Circuits of the Mind. *Oxford University Press.*

C. von der Malsburg, W. Schneider. (1986) A neural cocktail-party processor. *Biol. Cybern.* 54: 29-40.

V. N. Vapnik, A. Y. Chervonenkis. (1971) On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications, vol. 16*, 264-280.

L. Watts. (1994) Event-driven simulation of networks of spiking neurons. *Advances in Neural Information Processing Systems, vol. 6, Morgan Kaufmann*: 927-934.