On Computation with Pulses

Wolfgang Maass, Berthold Ruf *

Abstract

We explore the computational power of formal models for computation with pulses. Such models are motivated by realistic models for biological neurons, and by related new types of VLSI ("pulse stream VLSI").

In preceding work it was shown that the computational power of formal models for computation with pulses is quite high if the pulses arriving at a computational unit have an approximately linearly rising or linearly decreasing initial segment. This property is satisfied by common models for biological neurons. On the other hand several implementations of pulse stream VLSI employ pulses that are approximately piecewise constant (i.e. step functions).

In this article we investigate the relevance of the shape of pulses in formal models for computation with pulses. It turns out that the computational power drops significantly if one replaces pulses with linearly rising or decreasing initial segments by piecewise constant pulses. We provide an exact characterization of the latter model in terms of a weak version of a random access machine (RAM). We also compare the language recognition capability of a recurrent version of this model with that of deterministic finite automata and Turing machines.

1 Introduction

Computations in biological neural systems employ pulses in order to transmit information between their computational units (i.e., neurons). For many decades one had thought that information is primarly encoded in the frequency of these pulses ("rate coding"). Under this assumption, computations in biological neural systems can be modeled quite well by common neural network models, that employ threshold gates or sigmoidal gates as computational units. More recent results have shown, however, that many biological neural systems encode information in the timing of individual pulses (see e.g. [Abeles 91], [Bialek 92], [Sejnowski 95], [Thorpe 89]). Thus the communication and also the "computation" of biological neurons in these systems is quite different from the way

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

in which processors in digital computers and also "neurons" in artificial neural networks operate.

In order to model such computations one has to resort to a quite different mathematical model for the computational activity of a biological neuron: the leaky integrate-and-fire neuron. Such neuron adds up incoming pulses in a quantity called membrane potential, which models the membrane potential at the "trigger-zone" (located at the beginning of the axon) of a biological neuron.

Whenever this membrane potential reaches a certain threshold, the neuron "fires" and emits a pulse (called actional potential or spike), which is transmitted through its axon via synapses to other neurons, where it causes another pulse: a "postsynaptic potential". If the firing of the presynaptic neuron causes an increase of the membrane potential in a postsynaptic neuron (and thus increases its chance to fire), then this postsynaptic potential is called an excitatory postsynaptic potential (EPSP). If the firing of the presynaptic neuron causes a decrease of the membrane potential, one speaks of an inhibitory postsynaptic potential (IPSP). In our mathematical model we will describe EPSP's and IPSP's at the trigger-zone of a neuron by socalled response-functions.

Computational units of this type can easily be built in hardware, and one has started to explore potential uses of this new mode of computation and communication in various VLSI-chips ("pulse stream VLSI") ([Murray 94], [Pratt 89], [Horinchi 91], [Jahnke 95], [Zaghloul 94]. But the principles and limitations of computations with artificial spiking neurons are so far only poorly understood.

One important task for the theoretical investigation of computations in formal models for networks of spiking neurons is to find out which aspects of the assumed model are *accidental* for its computational power, and which ones are *essential*. As part of this program we investigate in this article the computational effect of the shape of the pulses that reach the trigger-zone of a neuron. More precisely, we analyze the impact of the assumed shape of the response functions (i.e. of the postsynaptic potentials) on the computational power of an SNN (which is a formal model for a spiking neuron network).

It has been shown that SNNs with linearly rising or linearly decreasing initial segments in their response functions can perform with a small number of spikes basic operations on analog variables in temporal coding such as addition, subtraction, and multiplication with a constant (see [Maass 96a]). Also noisy versions of such networks turn out to be quite powerful ([Maass 97a]).

In this article we focus on SNNs whose response functions are described by piecewise constant functions (i.e. step-functions). This is certainly the simplest type of response functions from a mathematical point of view. In addition, such functions approximate quite well the shape of pulses that are currently employed in pulse stream VLSI. We show that the computational power of such SNNs is considerable weaker than that of a SNN with pulses that rise or drop linearly. They can no longer carry out the abovementioned operations on analog variables in temporal coding. Consequently there exists already a significant difference in the computational power of spiking neurons with response functions of the types (b) and (c) in Figure 1.

Besides our investigation of computations with numerical (i.e. analog) inputs, we also consider the case of computations of SNNs on bit strings, and show that their computational power drops from that of an arbitrary Turing machine to that of a deterministic finite automaton if their response functions are required to be piecewise constant instead of piecewise linear. In particular we will show that such SNNs can no longer carry out basic pattern matching operations in polynomial time.

Our negative results regarding the computational power of models with piecewise constant response functions hold even if there is no noise in the system. This should be contrasted with the positive results for models with piecewise linear response functions that hold even in the presence of noise (see e.g. [Maass 97a]).

Another important component of the common model for a biological neuron is its threshold function. Whereas a threshold gate has a static threshold, the firing threshold of a biological neuron varies over time in dependency of its recent firing history (hence we refer to it as a threshold *function*). In particular, a neuron has a higher threshold right after it has fired ("refractory period"). We consider in this article different types of threshold functions, and show that their shape has less influence on the computational power of the network than the shape of the response functions.

We review in section 2 of this article the precise models that are used. In section 3 we show that for numerical inputs and outputs the computational power of networks of spiking neurons with piecewise constant response functions can be characterized completely in terms of a conceptually very simple variation of the familiar random access machine ("N^--RAM"). We then use this characterization in Theorem 3.3 and Corollary 3.4 to derive the main results of this article. In section 4 we analyze the computational power of the here considered SNNs for digital computations, and prove that SNNs with piecewise constant response functions cannot carry out in polynomial time a simple pattern matching task, which can be carried out in linear time by SNNs with piecewise linear response functions.

An extended abstract of this article was presented at the ICANN'95 ([Maass 95d]). We refer to [Maass 96b], [Maass 97b] for further information and refernces.

2 Basic Definitions and Assumptions

In [Maass 96a] a rather general formal definition of a spiking neuron network (SNN) has been introduced, which allows the investigation of the computational power of different types of response- and threshold functions. We recall here this definition:

Definition 2.1 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 "neurons" and to the elements of E as "synapses")*
- *a subset $V_{in} \subseteq V$ of input neurons*
- *a subset $V_{out} \subseteq V$ of output neurons*
- *for each neuron $v \in V - V_{in}$ a threshold function $\Theta_v : \mathbf{R}^+ \to \mathbf{R} \cup \{\infty\}$ (where $\mathbf{R}^+ := \{x \in \mathbf{R} : x \geq 0\}$)*
- *for each synapse $\langle u, v\rangle \in E$ a response function $\varepsilon_{u,v} : \mathbf{R}^+ \to \mathbf{R}$ and a weight $w_{u,v} \in \mathbf{R}^+$.*

We assume that the firing of the input neurons $v \in V_{in}$ is determined from outside of $\mathcal{N}$, i.e. the sets $F_v \subseteq \mathbf{R}^+$ of firing times ("spike trains") for the neurons $v \in V_{in}$ are given as the input of $\mathcal{N}$.

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

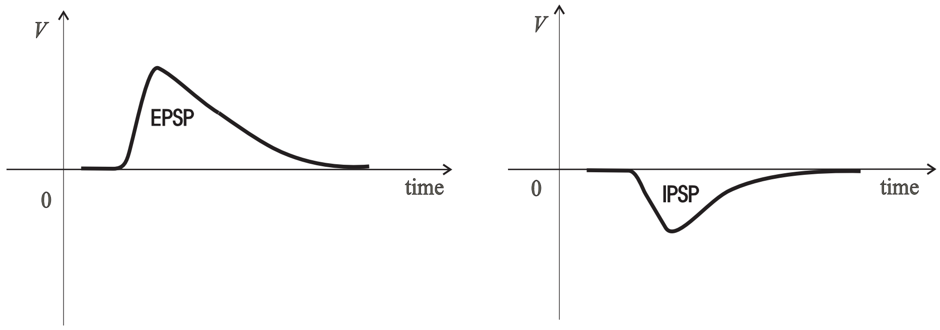
$$P_v(t) := 0 + \sum_{u\,:\,\langle u,v\rangle \in E} \; \sum_{s \in F_u\,:\,s<t} w_{u,v} \cdot \varepsilon_{u,v}(t-s)$$

(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 output of $\mathcal{N}$.

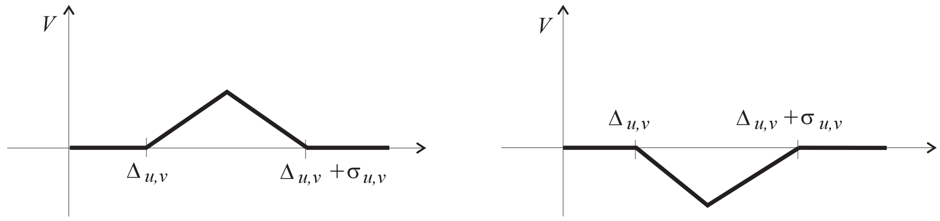
The complexity of a computation in an SNN is evaluated by counting each spike as a computation step.

This formal model is essentially a noise-free version of the *spike response model* as described in [Gerstner 91], [Gerstner 92] and [Gerstner 94]. One uses the response function $\varepsilon_{u,v}$ in order to describe the potential change or "postsynaptic potential" $w_{u,v} \cdot \varepsilon_{u,v}(t-s)$ at the trigger-zone of neuron v at time t, as a result of a firing of neuron u at time s. For simplicity the resting value of the membrane potential at the trigger-zone of neurons is normalized to 0.

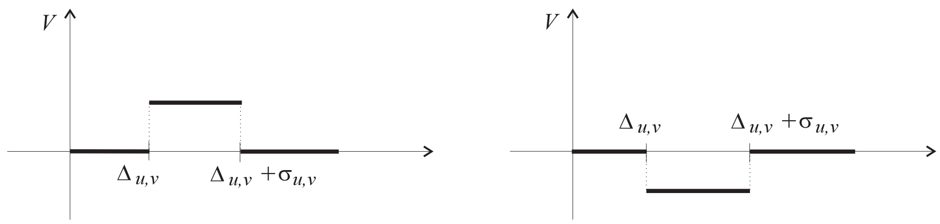
For the constructions in this article it suffices to make the following rather weak assumptions about the response- and threshold functions in an SNN.
All response functions $\varepsilon_{u,v} : \mathbf{R}^+ \to \mathbf{R}$ and threshold functions $\Theta_u : \mathbf{R}^+ \to \mathbf{R}^+ \cup \{\infty\}$ are some arbitrary functions with the following properties: There exist constants $\Delta_{min}, \Delta_{max}, \sigma_{min}, \sigma_{max} \in \mathbf{R}^+$ with $0 < \Delta_{min} < \Delta_{max}$ and $0 < \sigma_{min} < \sigma_{max}$ such that the following conditions are fulfilled:



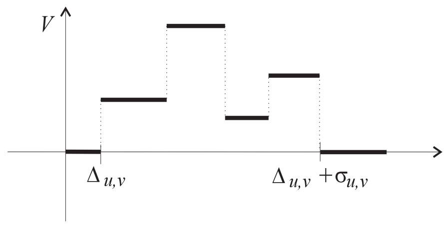
(a) Typical shape of EPSPs and IPSPs for biological neurons.



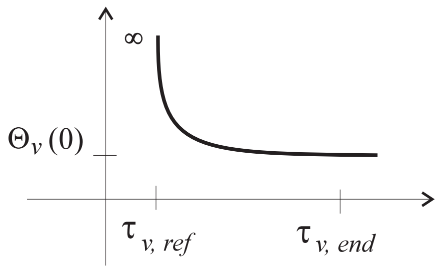
(b) Simple piecewise linear response functions $\varepsilon_{u,v}$.



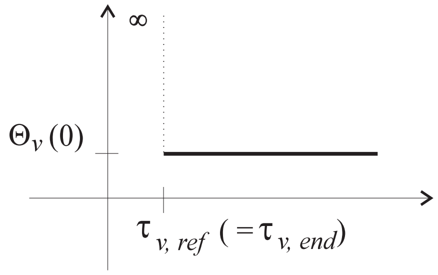
(c) Simple piecewise constant response functions $\varepsilon_{u,v}$.



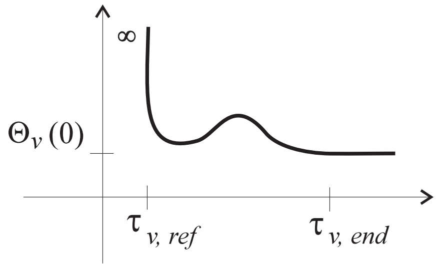
(d) Example for a complicated piecewise constant response function $\varepsilon_{u,v}$ that satisfies our conditions 1 - 3.



(e) Typical shape of the threshold function Θ_v for biological neurons v.



(f) Simple piecewise constant threshold function Θ_v.



(g) Example for a more complicated piecewise monotone and continuous threshold function Θ_v that satisfies our condition 4.

Figure 1: *Examples for response- and threshold functions.*

1. For every $\varepsilon_{u,v}$ there exists some delay $\Delta_{u,v} \in [\Delta_{min}, \Delta_{max}]$ and some $\sigma_{u,v} \in [\sigma_{min}, \sigma_{max}]$ such that $\varepsilon_{u,v}(x) = 0$ for all $x \in [0, \Delta_{u,v}] \cup [\Delta_{u,v} + \sigma_{u,v}, \infty)$.

2. Every $\varepsilon_{u,v}$ either satisfies $\varepsilon_{u,v}(x) > 0$ for all $x \in (\Delta_{u,v}, \Delta_{u,v} + \sigma_{u,v})$ (in which case we refer to it as an EPSP) or $\varepsilon_{u,v}(x) < 0$ for all $x \in (\Delta_{u,v}, \Delta_{u,v} + \sigma_{u,v})$ (in which case we refer to it as an IPSP).

3. For every EPSP-response function $\varepsilon_{u,v}$ there exists some $\varepsilon_{u,v}^{max} > 0$ with $\varepsilon_{u,v}^{max} = \max\{\varepsilon_{u,v}(x) : x \in (\Delta_{u,v}, \Delta_{u,v} + \sigma_{u,v})\}$. Furthermore for all $x_1, x_2 \in (\Delta_{u,v}, \Delta_{u,v} + \sigma_{u,v})$ with $\varepsilon_{u,v}(x_1) = \varepsilon_{u,v}(x_2) = \varepsilon_{u,v}^{max}$ it follows that $\varepsilon_{u,v}(x) = \varepsilon_{u,v}^{max}$ for all $x \in [x_1, x_2]$.

4. For every $v \in V - V_{in}$ there exist constants $\tau_{v,ref}$, $\tau_{v,end} \in \mathbf{R}^+$, such that $\Theta_v(x) = \infty$ for all $x \in (0, \tau_{v,ref})$ ("absolute refractory period"), $0 < \Theta_v(0) \leq \Theta_v(x)$ for all $x \in [\tau_{v,ref}$, $\tau_{v,end})$ ("relative refractory period"), and $\Theta_v(x) = \Theta_v(0) > 0$ for all $x \geq \tau_{v,end}$.

In this article we focus on the computational power of SNNs with piecewise constant response functions $\varepsilon_{u,v} : \mathbf{R}^+ \to \mathbf{R}$ and piecewise monotone and continuous (respectively piecewise linear) threshold functions $\Theta_u : \mathbf{R}^+ \to \mathbf{R}^+ \cup \{\infty\}$. When considering piecewise constant response functions we assume that for every $\varepsilon_{u,v}$ there exist constants $m_{u,v} \in \mathbf{N}$ and $t_1^{u,v}, \ldots, t_{m_{u,v}-1}^{u,v} \in \mathbf{R}^+ \backslash \{0\}$ with $t_i^{u,v} < t_{i+1}^{u,v}, 1 \leq i \leq m_{u,v} - 2$ such that the domain $\mathbf{R}^+$ of that function can be partitioned into $m_{u,v}$ intervals $[0, t_1^{u,v}), [t_i^{u,v}, t_{i+1}^{u,v})$ with $1 \leq i \leq m_{u,v} - 2$ and $[t_{m_{u,v}-1}^{u,v}, \infty)$ such that $\varepsilon_{u,v}$ is constant on each of these intervals. We will choose $t_1^{u,v} = \Delta_{u,v}$ and $t_{m_{u,v}-1}^{u,v} = \Delta_{u,v} + \sigma_{u,v}$. A piecewise constant threshold function Θ_u is defined in the same fashion (see Figure 1).

In biological models one usually assumes in addition that the sign of the derivative of each response function $\varepsilon_{u,v}$ changes only once. It turns out that our negative results (i.e. lower bound results) even hold for the larger class of models where this assumption is *not* imposed, and hence we do not make this assumption in our formal model.

In this article we are interested in relating the computational power of various kinds of SNNs to other computational models. We employ for that purpose the common notion of a *real-time simulation* from computational complexity theory (see e.g. [Leong 81], [Paul 84], [Reischuk 90], [Maass 96a]). One says that $\mathcal{M}'$ simulates $\mathcal{M}$ in real-time if $\mathcal{M}'$ can simulate each step in a computation of $\mathcal{M}$ with a *fixed* number of computation steps (i.e. the simulation of "later" computation steps of $\mathcal{M}$ does not require more steps of $\mathcal{M}'$ than the simulation of the first ones).

It is obvious that if $\mathcal{M}'$ simulates $\mathcal{M}$ in real-time, then it also simulates $\mathcal{M}$ in polynomial time (in fact: in linear time).

For biological neural systems the precise timing of computations is essential, and most computations are completed within a fixed number of "clock-cycles".

Hence the notion of a real-time simulation is better suited for their investigation than the more common but too coarse notion of a polynomial simulation.

When we say in the following that a class $\mathcal{C}$ of machines can be simulated in real-time by SNNs with response- and threshold functions of a certain type (e.g. piecewise constant), we mean the following: We can construct for any machine $\mathcal{M}$ in $\mathcal{C}$ an SNN $\mathcal{M}'$ that simulates $\mathcal{M}$ in real-time, where *we* choose the architecture of $\mathcal{M}'$ as well as the values of delays $\Delta_{u,v}$ and weights $w_{u,v}$ in $\mathcal{M}'$ and the "sign" of the response function (i.e. EPSP or IPSP). However we allow that the exact shape of the response- and threshold functions of $\mathcal{M}'$ is *given* to us, i.e. they can be *arbitrary* functions of the specified type (e.g. piecewise constant) that satisfy the conditions 1. - 4. specified above. Hence a simulation result of this type yields a real-time simulation of $\mathcal{M}$ with the simplest examples of such response- and threshold functions (see Figure 1, c and f), but also with any other response- and threshold functions that happen to satisfy the same conditions (see Figure 1, d and g). Thus a result of this type automatically implies that the exact shape of the response- and threshold functions of the considered type does not matter for this simulation result.

In the next section we will show that SNNs with piecewise constant response functions are real-time equivalent to a special type of random access machine (RAM), which we will call N^--RAM.

Definition 2.2 (N^--RAM)
An N^--RAM is a random access machine (RAM) with a constant number of registers that receives as its input, stores in its registers, operates on, and outputs real numbers from some bounded interval $[-B, B]$. The contents of a register R is denoted by $[R]$. The machine uses some designated register A as an accumulator. It can execute arbitrary programs of finite length, where each program statement has some unique label and consists of one of the following instructions:

ADD(β): *given some arbitrary constant $\beta \in [0, B]$, this command adds β to $[A]$ (provided that $[A] + \beta \in [-B, B]$) and stores the result in A.*

SUBTRACT(β): *given some arbitrary constant $\beta \in [0, B]$, this command subtracts β from $[A]$ (provided that $[A] - \beta \in [-B, B]$) and stores the result in A.*

IF COMPARE(R) THEN GOTO label: *this command compares the contents of the registers R and A. If $[A] \geq [R]$, then a jump to "label" is executed.*

GOTO label: *jumps to "label".*

LOAD(R): *loads the contents of R into A.*

STORE(R): *stores the contents of A into R.*

HALT: *unique instruction that ends the execution of the program.*

The input is given as the initial content of certain registers, and the output is given as the difference between the contents of two other distinguished registers when the machine halts. The complexity of a computation is evaluated according to the unit-cost criterion, where each execution of an instruction is counted as one computation step (regardless of the complexity of the operands).

Our output convention for N^--RAMs is motivated by the goal to prove that N^--RAMs and SNNs with piecewise constant response functions are real-time equivalent (Theorem 3.3). For SNNs we adopt the natural output convention that analog output values are represented as the difference between the firing times of two output neurons.

In [Maass 97a] the stronger model of an N-RAM had been considered that can in addition execute the instructions ADD, SUBTRACT (on *two* arbitrary reals from some bounded interval), and MULTIPLY(β). This machine model was shown in [Maass 95a] and [Maass 97a] to be real-time equivalent to SNNs with piecewise linear activation functions and to recurrent analog neural nets with piecewise linear activation functions.

We will use the adjectives *analog, numerical* and *real-valued* interchangeably throughout this article.

3 Characterization of the Power of Restricted SNNs for Analog Computations

Theorem 3.1 *Any SNN with piecewise constant response functions and piecewise monotone and continuous threshold functions can be simulated for real-valued input and output from a bounded range in real-time by an N^--RAM.*

Proof We will show that for any given SNN $\mathcal{N}$ of the here considered type one can construct an N^--RAM $\mathcal{M}$ that can simulate $\mathcal{N}$ in real-time. The basic idea of the proof is that given the firing time t of some neuron in $\mathcal{N}$, the simulating N^--RAM $\mathcal{M}$ computes for each neuron v in $\mathcal{N}$ the *potential firing time* $t_v > t$, i.e. the first time where v would fire provided that no other neuron fires within the time-interval (t, t_v). From Definition 2.1 it follows that the neuron v for which t_v is minimal actually fires (there might in fact be several neurons that fire simultaneously at time t_v).

$\mathcal{M}$ reserves for each neuron in $\mathcal{N}$ a fixed number of registers. The firing times of the input neurons of $\mathcal{N}$ are assumed to be given to $\mathcal{M}$ as input in the form of the initial content of some registers. For each firing time t of an input neuron and for each later firing time t that it has already "constructed", $\mathcal{M}$ proceeds as follows: We assume that $\mathcal{M}$ stores for each neuron u all its firing times $\leq t$ which can still be relevant at times $\geq t$ for some other neuron v, i.e. which occured within the time-interval $(t-\sigma_{max}-\Delta_{max}, t]$.[1] With $\tau_{min} := \min\{\tau_{u,ref} : u \in V\}$ it follows that at most $(\lfloor(\sigma_{max}+\Delta_{max})/\tau_{min}\rfloor + 1) \cdot |V|$ registers are needed for that, since a neuron u can generate in a time interval of length $\sigma_{max}+\Delta_{max}$ at most $\lfloor(\sigma_{max}+\Delta_{max})/\tau_{u,ref}\rfloor + 1$ spikes.

We now show how $P_v(t)$ can be computed by $\mathcal{M}$: Observe that $P_v(t)$ can assume only finitely many values, since v can receive only a bounded number of EPSPs respectively IPSPs which are still relevant at time t from some neuron u with $\langle u, v\rangle \in E$. With our assumption on the shape of the response functions it follows that each of them can contribute to $P_v(t)$ only one of finitely many values (of the form $w_{u,v} \cdot \varepsilon_{u,v}(t-t_u)$, with t_u being some firing time of neuron u).

In order to compute $P_v(t)$ at some time t, $\mathcal{M}$ has by definition to compute for every neuron u with $\langle u, v\rangle \in E$ and all firing times $t_u < t$ of u, which can be relevant to $P_v(t)$ the contribution $w_{u,v} \cdot \varepsilon_{u,v}(t-t_u)$ to $P_v(t)$. Therefore $\mathcal{M}$ has to find the largest $t_j^{u,v}$ (see Section 2) with $t_j^{u,v} \leq t - t_u$. Since $\varepsilon_{u,v}(t)$ is assumed to be piecewise constant, all possible values of $t_i^{u,v}$ and $w_{u,v} \cdot \varepsilon_{u,v}(t_i^{u,v})$ can be stored in a lookup-table (provided in the form of constants in certain registers of $\mathcal{M}$). If $j < m_{u,v} - 1$ then $\mathcal{M}$ also stores in some register the next time $t'_u = t_u + t_{j+1}^{u,v}$, when $P_v(t)$ changes again due to the firing of neuron u at t_u.

With the knowledge of $P_v(t)$ and of the first time t' when the potential changes after t (given by the minimum of all the t'_u for all u with $\langle u, v\rangle \in E$), $\mathcal{M}$ can easily check whether there is some potential firing time during the time-interval $[t, t')$, i.e. a time where the potential function P_v meets the threshold function Θ_v. Since P_v can assume only finitely many values, it is possible to store in a lookup-table for every v and for every possible value P of P_v all $\tilde{t}$ for which $\Theta_v(\tilde{t}) = P$. If Θ_v has some constant segment of value P, $\mathcal{M}$ simply stores the corresponding interval-boundaries. Note that there are only finitely many values of $\tilde{t}$ to be stored due to our assumptions about the threshold function. Using this table $\mathcal{M}$ can find out whether there exists some $\tilde{t}$, such that $t_v + \tilde{t}$ is within $[t, t')$ and whether $\Theta_v(\tilde{t}) = P_v(t_v + \tilde{t})$ with t_v being the last firing time of neuron v (if v has not fired before t then we set $t_v = 0$ and replace $\Theta_v(\tilde{t})$ by $\Theta_v(0)$). If such $\tilde{t}$ can be found, then the smallest $\tilde{t}$ fulfilling this condition yields the next potential firing time $t_v + \tilde{t}$ of v. Otherwise $\mathcal{M}$ has to check iteratively whether the threshold is exceeded within the next constant segment of P_v.

Obviously it is sufficient to search for the potential firing times of v within the time-interval $[t, t+\sigma_{max}+\Delta_{max})$ since no spike which occured before t can have any influence on v at times $\geq t + \sigma_{max} + \Delta_{max}$. As mentioned above the neuron with the smallest potential firing time actually fires. $\mathcal{M}$ stores this time

[1] It would be sufficient to consider the time-interval $(t - \sigma_{max,u} - \Delta_{max,u}, t]$ for every neuron u with $\Delta_{max,u} = \max\{\Delta_{u,v} : v \in V\}$ and $\sigma_{max,u} = \max\{\sigma_{u,v} : v \in V\}$, but this gain in efficiency is irrelevant for our proof.

in a register dedicated to this neuron and then continues its simulation in the same manner (taking this new firing time into account).

The registers containing (potential) firing times of $\mathcal{M}$ have to be kept bounded. Since only a bounded "time-window" of previous firings is relevant for determining future firings, M can subtract from the contents of all registers containing such firing times a suitable constant C and can erase those among these registers whose content is $< C$. ∎

Theorem 3.2 *Any N^--RAM can be simulated in real-time by an SNN with piecewise constant response- and threshold functions.*

PROOF We show that SNNs of the here considered type can store real numbers from some bounded interval with the help of oscillators, and that they can simulate within a bounded number of spikes every possible N$^-$-RAM instruction. The argument is based on a proof given in [Maass 96a] for the real-time simulation of Turing machines by a less restricted class of SNNs.

In our SNN model, *oscillators* can be realized using two neurons u and v with $\langle u, v\rangle \in E$, $\langle v, u\rangle \in E$, $\varepsilon_{u,v}$ and $\varepsilon_{v,u}$ being EPSP-response functions. The weights are chosen such that $w_{u,v} \geq \Theta_v(0)/\varepsilon^{max}_{u,v}$ and $w_{v,u} \geq \Theta_u(0)/\varepsilon^{max}_{v,u}$. Once started, a spike "cycles" periodically through these two neurons. Such an oscillator has two inputs with which the oscillation can be started respectively halted, and one output, through which spikes with the oscillation period are sent out. These oscillators can be used in two ways for storing data: They can be used for storing single bits using their two states oscillating/dormant. Assuming the existence of a designated oscillator, which we call the *pacemaker* PM with oscillation period π_{PM}, any other oscillator O with the same oscillation period can also be used for representing real numbers modulo π_{PM} as the phase difference between O and PM. In order to represent negative numbers we assume that each oscillator representing some real number is associated with a second oscillator O_s representing the sign of that number. Numbers greater equal zero are represented by O as described above where O_s is dormant, for negative numbers we assume that O_s is oscillating with the same frequency and phase difference to PM as O. Note that for arbitrary $a \in \mathbf{R}^+$ we get $(-a \equiv \pi_{PM} - a) \bmod \pi_{PM}$. In order to represent for the given constant B of a given N$^-$-RAM all possible register contents $\in [-B, B]$, we assume that $\pi_{PM} > B$.[2] The oscillator corresponding to the accumulator A as described in Definition 2.2 will be denoted with O_A.

The program control can be realized in the same way as in [Maass 96a], where it has been shown how SNNs can simulate arbitrary threshold circuits

[2] If B is greater than the sum of the lengths of the time-intervals it takes $\varepsilon_{u,v}$ and $\varepsilon_{v,u}$ to reach $\varepsilon^{max}_{u,v}$ respectively $\varepsilon^{max}_{v,u}$, then one has to use a cycle of more than two neurons.

with boolean input, and thus simulate the control of some Turing machine in a very efficient way. This construction can also be applied for the type of SNNs considered here. A given N^--RAM-program $\mathcal{P}$ can be described by a boolean function if we assume that each N^--RAM-program statement is associated with a certain unique state. Each state can be described in a binary way using oscillators where a "1" ("0") is represented by an oscillating (dormant) state. We will refer to these oscillators as "state-oscillators". For every N^--RAM-operation occuring in $\mathcal{P}$ we will construct one module which is started by the firing of some designated neuron, which acts as input neuron for that module. Thus there are as many ADD-modules as ADD-statements in $\mathcal{P}$ using different parameters (the same holds for the other parametrized statements). By using a layer of inter-neurons between the state-oscillators and those modules, it can be easily achieved that a certain state of the state-oscillators activates one unique module, executing the corresponding N^--RAM-operation.

We now show how the N^--RAM-operations can be realized on SNNs of the here considered type. For the COMPARE(R) operation we assume that the contents of R has been copied to some designated register R'. We use one neuron u that fires iff $[A] < [R']$. It receives input only from some inter-neurons u_1 and u_2 with equal delays, which receive within one period π_{PM} of PM at time t_1 and t_2 a spike from the oscillator representing $[R']$ respectively from O_A. We assume that the connection from u_1 to u is inhibitory, from u_2 to u excitatory, that $\Delta_{u_1,v} = \Delta_{u_2,v}$ and that $w_{u_1,u}$ and $w_{u_2,u}$ are chosen such that $\varepsilon_{u_1,u}(t) \cdot w_{u_1,u} < -\varepsilon^{max}_{u_2,u} \cdot w_{u_2,u}$ for all $t \in (\Delta_{u_1,u},\ \Delta_{u_1,u} + \sigma_{u_1,u})$ and $\varepsilon_{u_2,u}(t_1^{u_2,u}) \cdot w_{u_2,u} \geq \Theta_u(0)$. The firing of u_1 has to contribute to the potential of u at least $-\varepsilon^{max}_{u_2,u} \cdot w_{u_2,u}$ during the time-interval $(t_1 + \Delta_{u_1,u},\ t_1 + \Delta_{u_1,u} + \pi_{PM})$ (note that $|t_1 - t_2| < \pi_{PM}$). Therefore it might be necessary to add some inhibitory inter-neurons. Hence u fires if the EPSP from u_2 reaches u before the IPSP from u_1 (which occurs if $t_2 < t_1$, i.e. iff $|[A]| < |[R']|$. By using suitable inhibition and excitation modules which inhibit u from firing for a sufficiently long time in the case that $[R'] < 0$ and $[A] > 0$, respectively guarantee that u fires in the case that $[R'] > 0$ and $[A] < 0$ (one simply has to check the corresponding "sign-oscillators"), we achieve that u fires iff $[A] < [R']$. The N^--RAM uses the COMPARE operation only in connection with a conditional jump. This can be simulated in the SNN by representing the two possible subsequent states in the same way as the state-oscillators. The firing of u causes one of them to be copied to the state-oscillators. In case that u does not fire within a certain time-interval, it can be easily achieved that the other subsequent state is copied to the state oscillators.

The simulation of the operation ADD(β) can basically be achieved by "sending" the spike of O_A through a delay module with delay β, i.e. a chain of neurons having only EPSP-links such that the delay of a spike passing through this chain adds up to β. In order to compute the sign of the addition, the result of that operation is temporarily stored in some oscillator O. Depending on the input $[A]$, $O_{A,s}$ has to be started/halted, which can be easily realized using a COMPARE-module. Finally the "content" of O is copied to O_A. SUBTRACT(β) can be realized in the same fashion.

The simulation of the LOAD(R)-operation signals to the oscillator representing $[R]$ to send a spike through its output to O_A. The delays have to be chosen such that O_A will actually represent $[R]$. We assumed that each LOAD-operation occuring in $\mathcal{P}$ corresponds to a unique state of the state oscillators. The proper register can be addressed in the same fashion as the different states of the state-oscillators address different modules. If $[R] < 0$ the corresponding sign-oscillators have to be copied as well.

The STORE(R)-operation can be realized in an analogous way by first halting the oscillator containing $[R]$ and then copying the "contents" of O_A to this oscillator. ■

Theorems 3.1 and 3.2 immediately imply:

Theorem 3.3 *SNNs with piecewise constant response- and threshold functions are for computations with bounded real-valued input and output real-time equivalent to N^--RAMs.*

A closer look reveals that Theorems 3.1 and 3.2 together also imply the following result:

Corollary 3.4 *SNNs with piecewise constant response- and threshold functions are real-time equivalent to the class of SNNs with piecewise constant response functions and piecewise monotone and continuous threshold functions.*

It has been shown in [Maass 96a] that an SNN that has small linearly increasing and decreasing segments in its response functions can add and subtract arbitrary bounded real numbers and also multiply arbitrary real bounded numbers with a given real constant. However with piecewise constant response functions this is not possible, as the following Theorem shows.

Theorem 3.5 *No SNN with piecewise constant response functions and piecewise monotone and continuous threshold functions can carry out with a bounded number of spikes any of the operations ADD, SUBTRACT, MULTIPLY(β) (for any $\beta > 0$ with $\beta \neq 1$) on arbitrary small differences in firing times between neurons. This holds even if the simulating SNN may employ arbitrary real-valued parameters.*

Proof We prove by contradiction a slightly stronger result: no such SNN can *decide* with a bounded number of spikes for given (arbitrarily small) differences $a, b, c \geq 0$ in firing times between certain neurons whether $a + b = c$, $a - b = c$, or $a \cdot \beta = c$ (for any fixed $\beta > 0$ with $\beta \neq 1$). Assume that there exists an SNN of the considered type that solves any of these decision-problems with a bounded number of spikes. By Theorem 3.1 this implies that there exists an N^--RAM $\mathcal{M}$ that can solve this decision problem for arbitrarily small inputs $a, b, c \geq 0$ with a bounded number of (say: at most ℓ) computation steps. Consider first the case where $\mathcal{M}$ decides whether $a + b = c$.

All possible computations of length $\leq \ell$ of this N^--RAM $\mathcal{M}$ can be simulated by a decision tree T of depth ℓ with some rather special form of linear decision at its branching nodes. All register contents of $\mathcal{M}$ can be represented as a sum of at most one of the inputs a, b, c and a finite number of constants β. The linear decisions at the branching nodes of T represent applications of the instruction COMPARE to two sums of this type. Thus assuming that T contains m branching nodes, for every $1 \leq i \leq m$ the ith linear decision is of the form "$r_i + \gamma_i \geq s_i + \delta_i$", where $r_i, s_i \in \{a, b, c, 0\}$ and $\gamma_i, \delta_i \in \mathbf{R}^+$ are certain constants for this branching node.

Now we consider arbitrary inputs a, b, c with $0 < a < b < c < \epsilon/2$ with $\epsilon = \min\{|\gamma_i - \delta_i| : \gamma_i \neq \delta_i \text{ and } 1 \leq i \leq m\}$. It follows that each comparison "$r_i + \gamma_i \geq s_i + \delta_i$" in T holds for *all* such a, b, c if $\gamma_i > \delta_i$, and holds for *no* such a, b, c if $\gamma_i < \delta_i$ (if $\gamma_i = \delta_i$, its validity is predetermined by the pre-arranged order $a < b < c$). This results in a contradiction since the computation of T will arrive for all these inputs $\langle a, b, c \rangle$ at the same leaf of T (and hence give the same output), in spite of the fact that $a + b = c$ holds for some of these inputs, and does not hold for others.

Since $a - b = c$ holds if and only if $a = b + c$, the preceding argument automatically also covers the case of decisions "$a - b = c$". The argument for "$a \cdot \beta = c$" is analogous. ■

Corollary 3.6 *(see [Maass 96b]) There exists a circuit consisting of 3 threshold gates that cannot be simulated by any SNN with piecewise constant response functions with a bounded number of spikes.*

Proof Consider a threshold circuit that outputs 1 for inputs $x_1, x_2, x_3 \in [0, 1]$ if $x_1 + x_2 = x_3$, and 0 else. Obviously this can be achieved by a circuit with just 3 threshold gates: the circuit outputs 1 if and only if $x_1 + x_2 \geq x_3$ AND $x_1 + x_2 \leq x_3$.

The negative part of the claim follows from the proof of Theorem 3.5. ■

Corollary 3.7 *No SNN with piecewise constant response functions and piecewise monotone and continuous threshold functions is able to double through computations that involve at most a bounded number of spikes a difference in firing times between neurons, or a phase-difference between two oscillators (not even for arbitrarily small phase-differences).*

Proof This follows directly from Theorem 3.5 (consider the operation MULTIPLY(2)). ■

4 Characterization of the Power of Restricted SNNs for Digital Computations

In this section we consider the case where the SNN receives an input $\underline{w} \in \{0,1\}^*$ in an online fashion, i.e. bit by bit, where $\{0,1\}^*$ is the set of all binary strings of finite length. We allow that the SNN signals through the firing of a designated neuron v_{prompt} that it wants to receive the next input bit. If the next input bit is "1", a designated neuron $v_{in} \in V_{in}$ will fire with a certain given delay $\Delta \in \mathbf{Q}$ after the firing of v_{prompt}. If the next input bit is "0", v_{in} will not fire before the next firing of v_{prompt}.

The following Theorem provides a stark contrast to the result in [Maass 96a], where it was shown that SNNs with piecewise linear response- and threshold functions and rational parameters can simulate arbitrary Turing machines.

Theorem 4.1 *SNNs with piecewise constant response functions and piecewise linear threshold functions with rational parameters are for online boolean input real-time equivalent to DFAs.*

PROOF Assume that some SNN $\mathcal{N}$ as in the claim is given. Since $\mathcal{N}$ uses only rational parameters, the times t where the potential of some neuron changes or some neuron fires can be shown to be a multiple of some constant $\delta \in \mathbf{Q}$. We will represent the current state of all neurons of $\mathcal{N}$ at such time t as some state of the DFA $\mathcal{A}$ and compute by a transition-function the next state at time $t+\delta$.

The construction of the simulating DFA is carried out as follows: We will define a finite set of SNN-states in terms of the states of all neurons of $\mathcal{N}$ in such a way that every SNN-state has a unique successor-state. For that purpose it suffices if the state of some neuron v at time t contains the following information:

- the spiking history of v, given as the time difference between t and all firing times of v which can be still relevant to other neurons at times $\geq t$, i.e. all firing times of v which occured within $(t - \sigma_{max} - \Delta_{max},\ t]$.
- for every neuron u with $\langle u, v\rangle \in E$ and every firing time $t_u < t$ of u given by $t - t_u$ with $t - t_u \in (0,\ \sigma_{max} + \Delta_{max})$, the contribution of this spike to $P_v(t)$, i.e. $w_{u,v} \cdot \varepsilon_{u,v}(t - t_u)$ and furthermore the time-difference $t' - t$ between the smallest $t' > t$ (if any) and t with $\varepsilon_{u,v}(t' - t_u) \neq \varepsilon_{u,v}(t - t_u)$. The state of v also depends on the number i of the current segment of $\varepsilon_{u,v}$, with $1 \leq i \leq m_{u,v} - 1$ such that $t' - t_u = t^{u,v}_{i+1}$. If no t' exists then $i = m_{u,v}$.
- the current threshold $\Theta_v(t - t_v)$, where t_v is the last firing time of v. $t - t_v$ is given by the smallest element from the spiking history of v. If v has not fired before t or if $t - t_v \geq \tau_{v,end}$, then the current threshold is $\Theta_v(0)$.

As shown in the proof of Theorem 3.1, $P_v(t)$ can assume only finitely many values, which are in this case rational. For each possible value P of the potential function of some neuron v it follows that each t for which $P = \Theta_v(t)$ (if any) such that t is within a non-constant segment of Θ_v is rational, since the threshold functions were assumed to be piecewise linear. We denote all these possible times with $\theta_{v,1}, \ldots, \theta_{v,n_v}$ for some suitable constant $n_v \in \mathbf{N}$.

If a neuron $v \neq v_{in}$ fires at time t, then there has to be at least one neuron u, which fired at some time $t_u < t$ and which caused the "last jump" in the potential function of v before or at t. We can express t in terms of previous firing times: Either $t = t_u + t_i^{u,v}$ for some i (i.e. the threshold was exceeded at a "jump" of the potential function) or $t = t_v + \theta_{v,j}$ for some j with t_v being the last firing time of v (i.e. the threshold was exceeded during a non-constant segment of the potential function). Obviously the latter case cannot occur if v did not fire within $[t - \tau_{v,end},\ t)$. Finally we observe that if the input neuron v_{in} fires at time t, then $t = t' + \Delta$ for a firing time t' of the neuron v_{prompt}.

By induction it follows that for every neuron v any time t where v fires or its potential changes is rational and of the form

$$t = \sum_i n_i t_i + \sum_j \tilde{n}_j \theta_j + \hat{n} \cdot \Delta \tag{1}$$

with $n_i, \tilde{n_j}, \hat{n} \in \mathbf{N}$, t_i of the form $t_{\tilde{i}}^{\tilde{u},\tilde{v}}$ and Θ_j of the form $\Theta_{\tilde{v},\tilde{j}}$ with $\tilde{i}, \tilde{j} \in \mathbf{N}$ and $\tilde{u}, \tilde{v} \in V$. The last term of equation 1 takes into account the delay Δ of the input neuron v_{in} after the firing of v_{prompt} (as described at the beginning of this section). Now we can easily choose a constant $\delta \in \mathbf{Q}$ such that for any such t there exists some $n \in \mathbf{N}$ with $t = n \cdot \delta$.

The preceding analysis implies that it is sufficient to consider $\mathcal{N}$ only at times $t = n \cdot \delta$ and that every neuron and thus also $\mathcal{N}$ can assume at those times only finitely many states. We model every state s of $\mathcal{N}$ as described above by a state s' of the DFA $\mathcal{A}$. Those states of $\mathcal{N}$ where v_{prompt} fires, will be mimicked by states of the DFA $\mathcal{A}$ where it reads its next input bit. A "1"-input causes the DFA to assume a state reflecting an SNN state where v_{in} fires at the corresponding time. Since there exists according to the preceding construction for each state of $\mathcal{N}$ and each time δ a unique successor state of $\mathcal{N}$ at time $t + \delta$, we can define a corresponding transition function on states of $\mathcal{A}$ which allows $\mathcal{A}$ to simulate $\mathcal{N}$ for arbitrary online boolean input.

On the other hand a DFA can be simulated in real-time by an SNN of the here considered type in the same way as described in the proof of Theorem 3.2, since the simulation of boolean circuits on SNNs described there can be achieved using exclusively rational parameters. The states of the DFA are simulated by an array of oscillators in the SNN with binary states oscillating/dormant. ■

An SNN of the type considered in Theorem 4.1, but with *real-valued* parameters, is computationally more powerful than a DFA, as the following corollary indicates:

Corollary 4.2 *Any Turing machine $\mathcal{M}$ can be simulated by an SNN with piecewise constant response functions and threshold functions (although not in real-time)*

PROOF In order to prove this result we have to design a mechanism which allows a *fixed* size SNN to store and manipulate bit sequences of *arbitrary* length.

It is well known that any Turing machine $\mathcal{M}$ can be simulated (however not in real time) by a counter machine $\mathcal{M}'$, having no tapes but two counters (see e.g. [Hopcroft 79]). At each step $\mathcal{M}'$ can either increase or decrease one counter by one, or check if one counter is zero. An SNN can realize a counter with an oscillator O using the same idea as described in the proof of Theorem 3.2 by representing the current value of a counter as the phase-difference between O and some pace-maker π_{PM}. We choose a suitable constant β such that $k \cdot \beta = l \cdot \pi_{PM}$ for any $k, l \in \mathbf{N}$ implies that $k = l = 0$. Now the SNN can realize a counter incrementation (respectively decrementation) by using the ADD(β) and SUBTRACT(β) modules, as described in the proof of Theorem 3.2. In order to check if the counter is zero one can use the same idea as for the COMPARE-module. ■

The preceding result shows that SNNs with piecewise constant response- and threshold functions can simulate arbitrary Turing machines, as it has been shown before for SNNs with response functions that contain linearly increasing and decreasing segments ([Maass 96a]). However our next result exhibits an important difference between both classes of SNNs with regard to *speed* of these simulations. Whereas with the latter class of SNNs one can simulate arbitrary Turing machines in *real-time* (hence in *linear time*), *no* polynomial time simulation is possible if the response functions are piecewise constant.

Theorem 4.3 *Assume that a language $L \subseteq \{0,1\}^*$ is accepted in polynomial time by some online SNN $\mathcal{N}$ with arbitrary piecewise constant response functions and arbitrary piecewise monotone and continuous threshold functions, whose definition may involve arbitrary <u>real-valued</u> parameters. Then for every $n \in \mathbf{N}$ the initial segment $L \cap \{0,1\}^{\leq n}$ of L can be accepted by some DFA with at most polynomially in n many states.*

PROOF Theorem 3.3 also holds for online SNNs: One simply has to consider online N^--RAMs, which have in addition to the N^--RAM introduced in Definition 2.2 a READ-command, causing the next input-bit to be stored into some designated register.

An on-line N^--RAM $\mathcal{M}$ which simulates the given SNN $\mathcal{N}$ in real-time accepts $L \cap \{0,1\}^{\leq n}$ in at most polynomially in n many steps. The program of $\mathcal{M}$ is by definition of finite length and thus uses a finite number of constants. The possibilities of $\mathcal{M}$ to change the contents of registers are very limited (it can basically only add or subtract constants). Each of the (say k) registers of $\mathcal{M}$ can assume within polynomially in n many steps at most $p(n)$ different values for some polynomial p, independent from the input. Hence the registers of $\mathcal{M}$ can assume at most $p(n)^k$ "states" within polynomially in n many steps. Therefore a DFA with polynomially in n many states can simulate $\mathcal{M}$ for inputs up to length n, and hence accept $L \cap \{0,1\}^{\leq n}$. ■

Corollary 4.4 *No SNN of the type considered in Theorem 4.3 can decide in polynomial time whether $\underline{w} = \underline{\tilde{w}}$ for two sequentially presented bit strings $\underline{w}, \underline{\tilde{w}} \in \{0,1\}^n$ (i.e. $\underline{w}\,\underline{\tilde{w}}$, or $\underline{w}\#\underline{\tilde{w}}$ with a separation marker $\#$, is given as input in an online fashion).*

Proof By Theorem 4.3 it is sufficient to consider some DFA which carries out such a decision for a fixed n. It can be easily shown that such a DFA has to employ at least 2^n states to record the first half $\underline{w}$ of the input. ■

The pattern matching task from Corollary 4.4 can obviously be carried out by a Turing machine in linear time. Hence no SNN of the type considered in Theorem 4.3 and Corollary 4.4 can simulate an arbitrary Turing machine in polynomial time (i.e. in such a way that the simulation of t Turing machine steps requires at most polynomially in t many spikes). This provides a strong contrast to the situation for SNNs with linearly increasing and decreasing segments in their response functions, that can simulate any Turing machine in real-time (hence in linear time) even if all their parameters are rationals.

5 Conclusion

We have shown that both for numerical and boolean inputs a model for a network of spiking neurons with piecewise constant pulses (i.e. response functions) has much less computational power than a model whose pulses have linearly increasing or decreasing segments. In addition, Theorem 3.3 provides a complete characterization of the computational power of such networks with piecewise constant pulses in terms of a mathematically very perspicuous (and easy to program) computational model: the N^--RAM.

On the side we have shown that the shape of the threshold functions has much less influence on the computational power of the network than the shape of the pulses.

Acknowledgement: We would like to thank Eric Allender and Pekka Orponen for helpful comments.

References

[Abeles 91] M. Abeles. (1991) Corticonics: Neural Circuits of the Cerebral Cortex. *Cambridge University Press.*

[Bair 94] W. Bair, C. Koch, W. Newsome, K. Britten. (1994) Reliable temporal modulation in cortical spike trains in the awake monkey. *Proc. of the Symposium on Dynamics of Neural Processing, Washington, USA.*

[Bialek 92] W. Bialek, F. Rieke. (1992) Reliability and information transmission in spiking neurons. *Trends in Neuroscience, vol. 15*, 428-434.

[Gerstner 91] W. Gerstner. (1991) Associative memory in a network of "biological" neurons. *Advances in Neural Information Processing Systems, vol. 3, Morgan Kaufmann*: 84-90.

[Gerstner 92] W. Gerstner, R. Ritz, J. L. van Hemmen. (1992) A biologically motivated and analytically soluble model of collective oscillations in the cortex. *Biol. Cybern.* 68: 363-374.

[Gerstner 94] W. Gerstner, J. L. van Hemmen. (1994) How to describe neuronal activity: spikes, rates, or assemblies? *Advances in Neural Information Processing Systems, vol. 6, Morgan Kaufmann (San Mateo)*, 463-470.

[Hopcroft 79] J. E. Hopcroft, J. D. Ullman. (1979) Introduction to automata theory, languages and computation. *Addison-Wesley Publishing Company Inc.*

[Hopfield 95] J. J. Hopfield. (1995) Pattern recognition computation using action potential timing for stimulus representations. *Nature, vol. 376*, 33-36.

[Horinchi 91] T. Horinchi, J. Lazzaro, A. Moore, C. Koch. (1991) A delay-line based motion detection chip. *Advances in Neural Information Processing Systems, vol. 3, Morgan Kaufmann (San Mateo)*, 406-412.

[Jahnke 95] A. Jahnke, U. Roth, H. Klar. (1995) Towards efficient hardware for spike-processing neural networks. *Proc. of the World Congress on Neural Networks, July 1995 in Washington.*

[Leong 81] B. Leong, J. Seiferas. (1981) New real-time simulations of multihead tape units. *J. of the ACM* 28, 166-180.

[Maass 95a] W. Maass. (1995) On the computational complexity of networks of spiking neurons (extended abstract). *Advances in Neural Information Processing Systems, vol. 7, MIT-Press*, 183-190.

[Maass 95b] W. Maass and B. Ruf. (1995) On the relevance of the shape of postsynaptic potentials for the computational power of spiking neurons. *Proc. of the International Conference on Artificial Neural Networks (ICANN), Paris, vol. 2*, 515-520.

[Maass 96a] W. Maass. (1996) Lower bounds for the computational power of networks of spiking neurons. *Neural Computation, vol. 8, issue 1*, 1-40.

[Maass 96b] W. Maass, "Networks of spiking neurons: the third generation of neural network models", *Neural Networks*, 1997, to appear. FTP-host:

archive.cis.ohio-state.edu, FTP-filename: /pub/neuroprose/maass.third-generation.ps.Z.

[Maass 97a] W. Maass (1997) Fast sigmoidal networks via spiking neurons. *Neural Computation, vol. 9*, 279–304.

[Maass 97b] On the relevance of time in neural computation and learning. Invited paper for the 8th International Conference on Algorithmic Learning Theory in Sendai (Japan), *Springer (Berlin)*, 1997.

[Murray 94] A. Murray, L. Tarassenko. (1994) Analogue Neural VLSI: A Pulse Stream Approach. *Chapman & Hall*.

[Paul 84] W. Paul. (1984) On heads versus tapes. Theoretical Computer Science 28, 1-12.

[Pratt 89] G. A. Pratt. (1989) Pulse Computation. *Phd-dissertation, MIT, Dept. of Elect. Eng. and Comp. Sci.*

[Reischuk 90] K. R. Reischuk. (1990) Einführung in die Komplexitätstheorie. Teubner (Stuttgart).

[Sejnowski 95] T. J. Sejnowski. (1995) Time for a new neural code? *Nature, vol. 376*, 21-22.

[Thorpe 89] S. J. Thorpe, M. Imbert. (1989) Biological constraints on connectionist modelling. *In: Connectionism in Perspective, Pfeifer, R., Schreter, Z., Fogelman-Soulié, F., and Steels, L., eds., Elsevier (North-Holland).*

[Zaghloul 94] M. L. Zaghloul, J. L. Meador, R. W. Newcomb, eds., *"Silicon Implementations of Pulse Coded Neural Networks"*, Kluwer Academic Publishers, 1994.