Real-Time Computing Without Stable States:

A New Framework for Neural Computation Based on Perturbations

Wolfgang Maass+, Thomas Natschläger+ & Henry Markram*

+ Institute for Theoretical Computer Science, Technische Universität Graz; A-8010 Graz, Austria

* Brain Mind Institute, Ecole Polytechnique Federale de Lausanne, CH-1015 Lausanne, Switzerland

Wolfgang Maass & Thomas Natschlaeger
Institute for Theoretical Computer Science
Technische Universitaet Graz
Inffeldgasse 16b, A-8010 Graz, Austria
Tel: +43 316 873-5811
Fax: +43 316 873-5805
Email: maass@igi.tu-graz.ac.at, tnatschl@igi.tu-graz.ac.at

Henry Markram
Brain Mind Institute
Ecole Polytechnique Federale de Lausanne
CE-Ecublens, CH-1015 Lausanne, Switzerland
Tel:+972-89343179
Fax:+972-89316573
Email:Henry.Markram@weizmann.ac.il

Address for Correspondence:
Wolfgang Maass

A key challenge for neural modeling is to explain how a continuous stream of multi-modal input from a rapidly changing environment can be processed by stereotypical recurrent circuits of integrate-and-fire neurons in real-time. We propose a new computational model for real-time computing on time-varying input that provides an alternative to paradigms based on Turing machines or attractor neural networks. It does not require a task-dependent construction of neural circuits. Instead it is based on principles of high dimensional dynamical systems in combination with statistical learning theory, and can be implemented on generic evolved or found recurrent circuitry. It is shown that the inherent transient dynamics of the high-dimensional dynamical system formed by a sufficiently large and heterogeneous neural circuit may serve as universal analog fading memory. Readout neurons can learn to extract in real-time from the current state of such recurrent neural circuit information about current and past inputs that may be needed for diverse tasks. Stable internal states are not required for giving a stable output, since transient internal states can be transformed by readout neurons into stable target outputs due to the high dimensionality of the dynamical system. Our approach is based on a rigorous computational model, the liquid state machine, that unlike Turing machines, does not require sequential transitions between well-defined discrete internal states. It is supported, like the Turing machine, by rigorous mathematical results that predict universal computational power under idealized conditions, but for the biologically more realistic scenario of real-time processing of time-varying inputs. Our approach provides new perspectives for the interpretation of neural coding, for the design of experiments and data-analysis in neurophysiology, and for the solution of problems in robotics and neurotechnology.

Introduction

Intricate topographically organized feed-forward pathways project rapidly changing spatio-temporal information about the environment into the neocortex. This information is processed by extremely complex but surprisingly stereotypic microcircuits that can perform a wide spectrum of tasks (Shepherd, 1988, Douglas et al., 1998, von Melchner et al., 2000). The microcircuit features that enable this seemingly universal computational power, is a mystery. One particular feature, the multiple recurrent loops that form an immensely complicated network using as many as 80% of all the synapses within a functional neocortical column, has presented an intractable problem both for computational models inspired by current artificial computing machinery (Savage, 1998), and for attractor neural network models. The difficulty to understand computations within recurrent networks of integrate-and-fire neurons comes from the fact that their dynamics takes on a life of its own when challenged with rapidly changing inputs. This is particularly true for the very high dimensional dynamical system formed by a neural microcircuit, whose components are highly heterogeneous and where each neuron and each synapse adds degrees of freedom to the dynamics of the system.

The most common approach for modeling computing in recurrent neural circuits has been to try to take control of their high dimensional dynamics. Methods for controlling the dynamics of recurrent neural networks through adaptive mechanisms are reviewed in (Pearlmutter, 1995). So far one has not been able to apply these to the case of networks of spiking neurons. Other approaches towards modeling computation in biological neural systems are based on constructions of artificial neural networks that simulate Turing machines or other models for digital computation, see for example (Pollack, 1991), (Giles et al., 1992), (Siegelmann et al., 1994), (Hyoetyniemi, 1996), (Moore, 1998). Among these there are models, such as dynamical recognizers, which are capable of real-time computing on online input (in discrete time). None of these approaches has been demonstrated to work for networks of spiking neurons, or any more realistic models for neural microcircuits. It was shown in (Maass, 1996) that one also can construct recurrent circuits of spiking neurons that can simulate arbitrary Turing machines. But all of these approaches require synchronization of all neurons by a central clock, a feature that appears to missing in neural microcircuits. In addition they require the construction of particular recurrent circuits, and cannot be implemented by evolving or adapting a given circuit. Furthermore the results of (Maass et al., 1999) on the impact of noise on the computational power of recurrent neural networks suggest that all these approaches break down as soon as one assumes that the underlying analog computational units are subject to Gaussian or other realistic noise distributions. Attractor neural networks on the other hand allow noise robust computation, but their attractor landscape is in general hard to control, and they need to have a very large set of attractors in order to store salient information on past inputs (for example 1024 attractors in order to store 10 bits). In addition they are less suitable for real-time computing on rapidly varying input streams because of the time required for convergence to an attractor. Finally, none of these approaches allows that several real-time computations are carried out in parallel within the same circuitry, which appears to be a generic feature of neural microcircuits.

In this article we analyze the dynamics of neural microcircuits from the point of view of a readout neuron, whose task is to extract information and report results from a neural microcircuit to other circuits. A human observer of the dynamics in a neural microcircuit would be looking for clearly distinct and temporally stable features, such as convergence to attractors. We show that a readout neuron, that receives inputs from hundreds or thousands of neurons in a neural

microcircuit, can learn to extract salient information from the high dimensional transient states of the circuit, and can transform transient circuit states into stable readouts. In particular each readout can learn to define its own notion of equivalence of dynamical states within the neural microcircuit, and can then perform its task on novel inputs. This unexpected finding of "readout-assigned equivalent states of a dynamical system" explains how invariant readout is possible despite the fact that the neural microcircuit may never re-visit the same state. Furthermore we show that multiple readout modules can be trained to perform different tasks on the same state trajectories of a recurrent neural circuit, thereby enabling parallel real-time computing. We present the mathematical framework for a computational model that does not require convergence to stable internal states or attractors (even if they do occur), since information about past inputs is automatically captured in the perturbations of a dynamical system, i.e. in the continuous trajectory of transient internal states. Special cases of this mechanism were already reported in (Buonomano et al., 1995) and (Dominey et al., 1995). Similar ideas have been discovered independently by Herbert Jaeger (Jaeger, 2001) in the context of artificial neural networks.

Computing without Attractors

As an illustration for our general approach towards real-time computing consider a series of transient perturbations caused in an excitable medium (see (Holden et al., 1991)), for example a liquid, by a sequence of external disturbances ("inputs") such as wind, sound, or sequences of pebbles dropped into the liquid. Viewed as an attractor neural network, the liquid has only one attractor state – the resting state – and may therefore seem useless for computational purposes. However, the perturbed state of the liquid, *at any moment in time*, represents present as well as past inputs, potentially providing the information needed for an analysis of various dynamic aspects of the environment. In order for such a liquid to serve as a source of salient information about present and past stimuli without relying on stable states, the perturbations must be sensitive to saliently different inputs but non-chaotic. The manner in which perturbations are formed and maintained would vary for different types of liquids and would determine how useful the perturbations are for such "retrograde analysis". Limitations on the computational capabilities of liquids are imposed by their time-constant for relaxation, and the strictly local interactions and homogeneity of the elements of a liquid. Neural microcircuits, however, appear to be "ideal liquids" for computing on perturbations because of the large diversity of their elements, neurons and synapses (see (Gupta et al., 2000)), and the large variety of mechanisms and time constants characterizing their interactions, involving recurrent connections on multiple spatial scales ("loops within loops").

The foundation for our analysis of computations without stable states is a rigorous computational model: the *liquid state machine*. Two macroscopic properties emerge from our theoretical analysis and computer simulations as necessary and sufficient conditions for powerful real-time computing on perturbations: a *separation property, SP*, and an *approximation property, AP*.

SP addresses the amount of separation between the trajectories of internal states of the system that are caused by two different input streams (in the case of a physical liquid, *SP* could reflect the difference between the wave patterns resulting from different sequences of disturbances).

AP addresses the resolution and recoding capabilities of the readout mechanisms - more precisely its capability to distinguish and transform different internal states of the liquid into given target outputs (whereas *SP* depends mostly on the complexity of the liquid, *AP* depends mostly on the adaptability of the readout mechanism to the required task).

Liquid State Machines

Like the Turing machine (Savage, 1998), the model of a liquid state machine (LSM) is based on a rigorous mathematical framework that guarantees, under idealized conditions, *universal computational power*. Turing machines, however, have universal computational power for off-line computation on (static) discrete inputs, while LSMs have in a very specific sense universal computational power for real-time computing with fading memory on analog functions in continuous time. The input function $u(\cdot)$ can be a continuous sequence of disturbances, and the target output can be some chosen function $y(\cdot)$ of time that provides a real-time analysis of this sequence. In order for a machine M to map input functions of time $u(\cdot)$ to output functions $y(\cdot)$ of time, we assume that it generates, at every time t, an internal "liquid state" $x^M(t)$, which constitutes its current response to preceding perturbations, i.e., to preceding inputs $u(s)$ for $s \leq t$ (Figure 1). In contrast to the "finite state" of a finite state machine (or finite automaton) this liquid state consists of analog values that may change continuously over time. Whereas the state set and the state transition function of a finite state machine is in general constructed for a

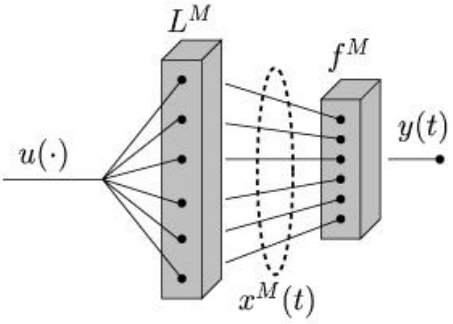

Figure 1: **A**: Architecture of a LSM. A function of time (time series) $u(\cdot)$ is injected as input into the liquid filter L^M, creating at time t the *liquid state* $x^M(t)$, which is transformed by a memory-less readout map f^M to generate an output $y(t)$.

specific task, the liquid states and the transitions between them need not be customized for a specific task. In a physical implementation this liquid state consists of all information about the current internal state of a dynamical system that is accessible to the readout modules. In

mathematical terms, this liquid state is simply the current output of some operator or filter[1] L^M that maps input functions $u(\cdot)$ onto functions $x^M(t)$:

$$x^M(t) = (L^M u)(t) \ .$$

In the following we will refer to this filter L^M often as *liquid filter*, or *liquid circuit* if implemented by a circuit. If it is implemented by a neural circuit, we refer to the neurons in that circuit as *liquid neurons*.

The second component of a LSM M is a memory-less readout map f^M that transforms, at every time t, the current liquid state $x^M(t)$ into the output

$$y(t) = f^M(x^M(t)) \, .$$

In contrast to the liquid filter L^M, this readout map f^M is in general chosen in a task-specific manner (and there may be many different readout maps, that extract different task-specific information in parallel from the current output of L^M). Note that in a finite state machine there exists no analogon to such task-specific readout maps, since there the internal finite states are already constructed in a task-specific manner. According to the preceding definition readout maps are in general memory-less[2]. Hence all information about inputs $u(s)$ from preceding time points $s \le t$ that is needed to produce a target output $y(t)$ at time t has to be contained in the *current* liquid state $x^M(t)$. Models for computation that have originated in computer science store such information about the past in stable states (for example in memory buffers or tapped delay lines). We argue, however, that this is not necessary since large computational power on functions of time can also be realized even if all memory traces are continuously decaying. Instead of worrying about the code and location where information about past inputs is stored, and how this information decays, it is enough to address the separation question: For which later time points t will any two significantly different input functions of time $u(\cdot)$ and $v(\cdot)$ cause significantly different liquid states $x_u^M(t)$ and $x_v^M(t)$. Good separation capability, in combination

[1] Functions F that map input functions of time $u(\cdot)$ on output functions $y(\cdot)$ of time are usually called operators in mathematics, but are commonly referred to as filters in engineering and neuroscience. We use the term *filter* in the following, and we write $(Fu)(t)$ for the output of the filter F at time t when F is applied to the input function $u(\cdot)$. Formally, such filter F is a map from U^n into $(\mathbf{R}^{\mathbf{R}})^k$, where $\mathbf{R}^{\mathbf{R}}$ is the set of all real-valued functions of time, $(\mathbf{R}^{\mathbf{R}})^k$ is the set of vectors consisting of k such functions of time, U is some subset of $\mathbf{R}^{\mathbf{R}}$, and U^n is the set of vectors consisting of n functions of time in U.

[2] The term "memory-less" refers to the fact that the readout map f^M is not required to retain any memory of previous states $x^M(s)$, $s < t$, of the liquid. However, in a biological context, the readout map will in general be subject to plasticity, and may also contribute to the memory capability of the system. We do not explore this issues in this article because the differentiation into a memory-less readout map and a liquid that serves as a memory device is made for conceptual clarification, and is not essential to the model.

with an adequate readout map f^M, allows us to discard the requirement of storing bits "until further notice" in stable states of the computational system.

Universal Computational Power of LSMs for Time Varying Inputs

We say that a class of machines has *universal power for computations with fading memory on functions of time* if any filter F, i.e., any map from functions of time $u(\cdot)$ to functions of time $y(\cdot)$, that is time invariant[3] and has fading memory[4], can be approximated by machines from this class, to any degree of precision. Arguably, these filters F are approximated according to this definition include all maps from input functions of time to output functions of time that a behaving organism might need to compute.

A mathematical theorem (see Appendix A) guarantees that LSMs have this universal computational power regardless of specific structure or implementation, provided that two abstract properties are met: the class of basis filters from which the liquid filters L^M are composed satisfies the point-wise separation property and the class of functions from which the readout maps f^M are drawn, satisfies the approximation property. These two properties provide the mathematical basis for the separation property *SP* and the approximation property *AP* that were previously discussed. Theorem 1 in Appendix A implies that there are no serious a priori limits for the computational power of LSMs on continuous functions of time, and thereby provides a theoretical foundation for our approach towards modeling neural computation. In particular, since this theorem makes no specific requirement regarding the exact nature or behaviour of the basis filters, as long as they satisfy the separation property (for the inputs in question), it provides theoretical support for employing instead of circuits that were constructed

---

[3] A filter F is called *time invariant* if any temporal shift of the input function $u(\cdot)$ by some amount t_0 causes a temporal shift of the output function $y = Fu$ by the same amount t_0, i.e., $(Fu^{t_0})(t) = (Fu)(t + t_0)$ for all $t, t_0 \in \mathbf{R}$, where $u^{t_0}(t) := u(t + t_0)$. Note that if U is closed under temporal shifts, then a time invariant filter $F : U^n \to (\mathbf{R}^{\mathbf{R}})^k$ can be identified uniquely by the values $y(0) = (Fu)(0)$ of its output functions $y(\cdot)$ at time 0.

[4] *Fading memory* (Boyd at al., 1985) is a continuity property of filters F which demands that for any input function $u(\cdot) \in U^n$ the output $(Fu)(0)$ can be approximated by the outputs $(Fv)(0)$ for any other input functions $v(\cdot) \in U^n$ that approximate $u(\cdot)$ on a sufficiently long time interval $[-T, 0]$. Formally one defines that $F : U^n \to (\mathbf{R}^{\mathbf{R}})^k$ has fading memory if for every $u \in U^n$ and every $\varepsilon > 0$ there exist $\delta > 0$ and $T > 0$ so that $\|(Fv)(0) - (Fu)(0)\| < \varepsilon$ for all $v \in U^n$ with $\|u(t) - v(t)\| < \delta$ for all $t \in [-T, 0]$. Informally a filter F has fading memory if the most significant bits of its current output value $(Fu)(0)$ depend just on the most significant bits of the values of its input function $u(\cdot)$ from some finite time window $[-T, 0]$ into the past. Thus, in order to compute the most significant bits of $(Fu)(0)$ it is not necessary to know the *precise* value of the input function $u(s)$ for any time s, and it is also not necessary to know *anything* about the values of $u(\cdot)$ for more than a finite time interval back into the past. Note that a filter that has fading memory is automatically causal.

for a specific task, partially evolved or even rather arbitrary "found" computational modules for purposeful computations. This feature highlights an important difference to computational theories based on Turing machines or finite state machines, which are often used as conceptual basis for modeling neural computation.

The mathematical theory of LSMs can also be extended to cover computation on spike trains (discrete events in continuous time) as inputs. Here the i^{th} component $u_i(\cdot)$ of the input $u(\cdot)$ is a function that assumes only the values 0 and 1, with $u_i(t)=1$ if the i^{th} preceding neuron fires at time t. Thus $u_i(\cdot)$ is not a continuous function but a sequence of point events. Theorem 2 in Appendix A provides a theoretical foundation for approximating any biologically relevant computation on spike trains by LSMs.

Neural Microcircuits as Implementations of LSMs

In order to test the applicability of this conceptual framework to modeling computation in neural microcircuits, we carried out computer simulations where a generic recurrent circuit of integrate-and-fire neurons (see Appendix B for details) was employed as liquid filter. In other words: computer models for neural microcircuits were viewed as implementation of the liquid filter L^M of an LSM. In order to test the theoretically predicted universal real-time computing capabilities of these neural implementations of LSMs, we evaluated their performance on a wide variety of challenging benchmark tasks. The input to the neural circuit was via one or several input spike trains, which diverged to inject current into 30% randomly chosen "liquid neurons". The amplitudes of the input synapses were chosen from a Gaussian distribution, so that each neuron in the liquid circuit received a slightly different input (a form of topographic injection). The liquid state of the neural microcircuit at time *t* was defined as all information that a readout neuron could extract at time *t* from the circuit, i.e. the output at time t of all the liquid neurons represented the current liquid state of this instantiation of a LSM. More precisely, since the readout neurons were modeled as I&F neurons with a biologically realistic membrane time constant of 30 ms, the liquid state $x^M(t)$ at time t consisted of the vector of contributions of all the liquid neurons to the membrane potential at time t of a generic readout neuron (with unit synaptic weights). Mathematically this liquid state $x^M(t)$ can be defined as the vector of output values at time *t* of linear filters with exponential decay (time constant 30 ms) applied to the spike trains emitted by the liquid neurons.

Each readout map f^{M} was implemented by a separate population P of integrate-and-fire neurons (referred to as "readout neurons") that received input from all the "liquid neurons", but had no lateral or recurrent connections[5]. The current firing activity *p(t)* of the population *P*, that is the fraction of neurons in P firing during a time bin of 20ms , was interpreted as the analog output of f^{M} at time t (one often refers to such representation of analog values by the current firing activity in a pool of neurons as *space rate coding*). Theoretically the class of readout maps that can be implemented in this fashion satisfies the approximation property *AP* (Maass, 2000, Auer et al., 2001), and is according to Theorem 1 in principle sufficient for approximating arbitrary given fading memory filters *F*. In cases where a readout with discrete values *1* and *0*

[5] For conceptual purposes we separate the "liquid" and "readout" elements in this paper, although dual liquid-readout functions can also be implemented.

suffices, one can implement a readout map even by a single I&F neuron that represents these discrete output values by firing/non-firing at time t. In cases where the target output consists of slowly varying analog values, a single readout neuron can be trained to represent these values through its time-varying firing rate. In any case the readout neurons can be trained to perform a specific task by adjusting the strengths of synapses projected onto them from the liquid neurons using a perceptron-like local learning rule (Auer et al., 2001). The final learned state of the readout neurons enables them to take particular weighted sums of the current outputs $x^M(t)$ of the liquid neurons and generate a response $f^M(x^M(t))$ that approximates the target value *y(t).*

As a first test of these neural implementations of LSMs we evaluated the separation property *SP* of computer models for neural microcircuits on spike train inputs. A large set of pairs of Poisson spike trains $u(\cdot)$ and $v(\cdot)$ were randomly generated and injected (in separate trials) as input to the recurrent neural circuit. The resulting trajectories $x_u^M(\cdot)$ and $x_v^M(\cdot)$ of liquid states of the recurrent circuit were recorded for each of these time-varying inputs $u(\cdot)$ and $v(\cdot)$. The average distance $\left\| x_u^M(t) - x_v^M(t) \right\|$ between these liquid states was plotted in Figure 2 as a function of the time t after the onset of the input, for various fixed values of the distance $d(u,v)$[6] between the two spike train inputs u and v. These curves show that the distance between these liquids states is well above the noise level, i.e. above the average liquid state differences caused by the same spike train applied with two different randomly chosen initial conditions of the circuit (indicated by the solid curve). Furthermore these curves show that the difference in liquid states is after the first 30 ms roughly proportional to the distance between the corresponding input

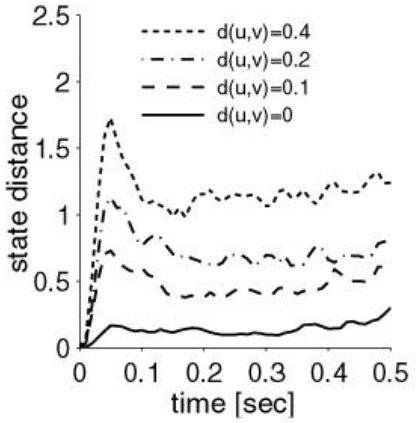

Figure 2: Average distance of liquid states for two different input spike trains u and v (given as input to the neural circuit in separate trials, each time with an independently chosen random initial state of the neural circuit, see Appendix B) plotted as a function of time t. The state distance increases with the distance *d(u,v)* between the two input spike trains u and v. Plotted on the y-axis is the average value of $\left\| x_u^M(t) - x_v^M(t) \right\|$, where ||.|| denotes the Euclidean norm, and $x_u^M(t)$, $x_v^M(t)$ denote the liquid states at time t for input spike trains u and v. The plotted results for the values 0.1, 0.2, 0.4 of the input difference d' represent the average over 200 randomly generated pairs u and v of spike trains such that $\left| d' - d(u,v) \right| < 0.01$. Parameters of the liquid: 1 column, degree of connectivity $\lambda = 2$ (see Appendix B for details).

[6] In order to define the distance $d(u,v)$ between two spike trains u and v we replaced each spike by a Gaussian $\exp(-(t/\tau)^2)$ for τ = 5ms (to be precise, u and v are convolved with the Gaussian kernel $\exp(-(t/\tau)^2)$) and defined $d(u,v)$ as the distance of the resulting two continuous functions in the L_2-norm (divided by the maximal lengths 0.5 s of the spike trains u and v).

spike trains. Note in particular the absence of chaotic effects for these generic neural microcircuit models with biologically realistic intermediate connection lengths.

Exploring the Computational Power of Models for Neural Microcircuit

As a first test of its computational power this simple generic circuit was applied to a previously considered classification task (Hopfield & Brody, 2001), where spoken words were represented by noise-corrupted spatio-temporal spike patterns over a rather long time interval (40-channel spike patterns over 0.5s). This classification task had been solved in (Hopfield & Brody, 2001) by a network of neurons designed for this task (relying on unknown mechanisms that could provide smooth decays of firing activity over longer time periods, and apparently requiring substantially larger networks of I&F neurons if fully implemented with I&F neurons). The architecture of that network, which had been customized for this task, limited its classification power to spike trains consisting of a single spike per channel.

We found that the same, but also a more general version of this spatio-temporal pattern recognition task that allowed several spikes per input channel, can be solved by a generic recurrent circuit as described in the previous section. Furthermore the output of this network was available at *any time*, and was usually correct as soon as the liquid state of the neural circuit had absorbed enough information about the input (the initial value of the correctness just reflects the initial guess of the readout). Formally we defined the correctness of the neural readout at time s by the term $1 - |$ target output $y(s) -$ readout activity $p(s) |$, where the target output $y(s)$ consisted in this case of the constant values 1 or 0, depending on the input pattern. Plotted in Fig. 3 is for any time t during the presentation of the input patterns in addition to the correctness as a function of t also the certainty of the output at time t, which is defined as the average correctness up to that time t. Whereas the network constructed by Hopfield and Brody was constructed to be invariant with regard to linear time warping of inputs (provided that only one spike arrives in each channel), the readouts of the generic recurrent circuit that we considered could be trained to be invariant with regard to a large class of different types of noises. The results shown in Fig. 3 are for a noise where each input spike is moved independently by an amount drawn from a Gaussian distribution with mean 0 and SD 32 ms.

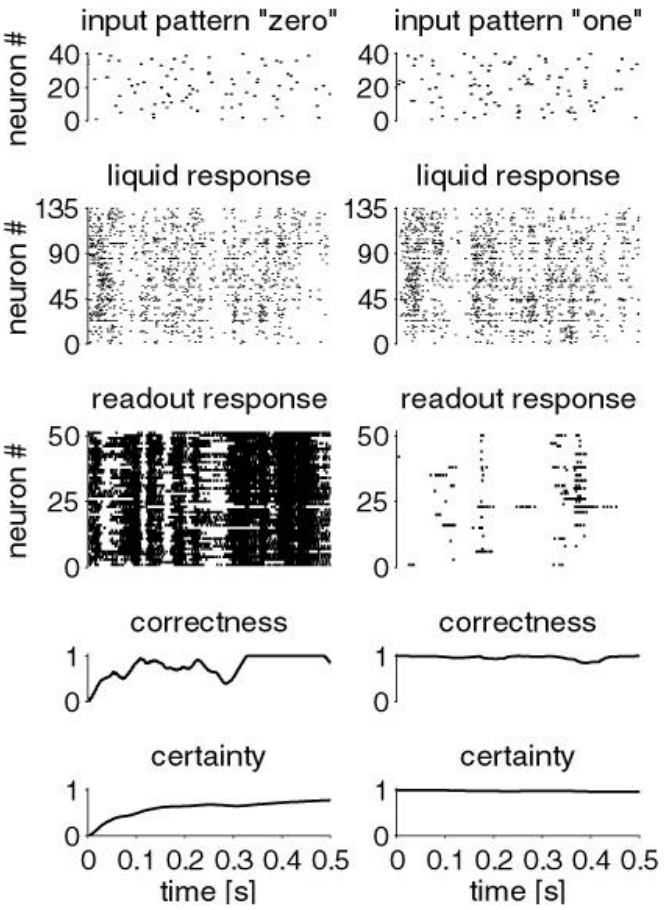

Figure 3: Application of a generic recurrent network of I&F neurons – modeled as LSM – to a more difficult version of a well-studied classification task (Hopfield & Brody, 2001). Five randomly drawn patterns (called "zero", "one ", "two", ..), each consisting of 40 parallel Poisson spike trains over 0.5s, were chosen. Five readout modules, each consisting of 50 integrate-and-fire neurons, were trained with 20 noisy versions of each input pattern to respond selectively to noisy versions of just one of these patterns (noise was injected by randomly moving each spike by an amount drawn independently from a Gaussian distribution with mean 0 and variance 32ms; in addition the initial state of the liquid neurons was chosen randomly at the beginning of each trial). The responses of the readout which had been trained* to detect the pattern "zero" is shown for a new, previously not shown, noisy versions of two of the input patterns. The correctness and certainty (= average correctness so far) are shown as functions of time from the onset of the stimulus at the bottom. The correctness is calculated as $1-|p(t)-y(t)|$ where $p(t)$ is the normalized firing activity in the readout pool (normalized to the range [0 1]; 1 corresponding to an activity of 180Hz; binwidth 20ms) and $y(t)$ is the target output. (Correctness starts at a level of 0 for pattern "zero" where this readout pool is supposed to become active, and at a value of 1 for pattern "one", because the readout pool starts in an inactive state). In contrast to most circuits of spiking neurons that have been constructed for specific computational task, the spike trains of liquid and readout neurons shown in this figure look rather "realistic".

*The familiar delta-rule was applied or not applied to each readout neuron, depending on whether the current firing activity in the readout pool was too high, too low, or about right, thus requiring at most two bits of global communication. The precise version of the learning rule was the p-delta rule that is discussed in Auer et al., (2001).

Giving a constant output for a time-varying liquid state (caused by a time-varying input) is a serious challenge for a LSM, since it cannot rely on attractor states, and the memory-less readout has to transform the transient and continuously changing states of the liquid into a stable output (see the discussion below and Fig. 9 for details). In order to explore the limits of this simple neural implementation of a LSM for computing on time-varying input, we chose another classification task where *all* information of the input is contained in its temporal evolution, more precisely in the interspike intervals of a single input spike train. In this test, 8 randomly generated Poisson spike trains over 250 ms, or equivalently 2 Poisson spike trains over 1000 ms partitioned

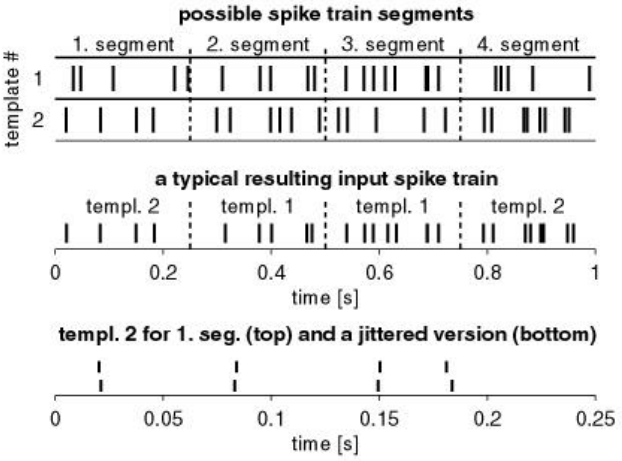

Figure 4: Evaluating the fading memory of a generic neural microcircuit: the task. In this more challenging classification task all spike trains are of length 1000 ms and consist of 4 segments of length 250 ms each. For each segment 2 templates were generated randomly (Poisson spike train with a frequency of 20 Hz); see upper traces. The actual input spike trains of length 1000 ms used for training and testing were generated by choosing for each segment one of the two associated templates, and then generating a noisy version by moving each spike by an amount drawn from a Gaussian distribution with mean 0 and a SD that we refer to as "jitter" (see lower trace for a visualization of the jitter with an SD of 4 ms). The task is to output with 4 different readouts at time t = 1000 ms for each of the preceding 4 input segments the number of the template from which the corresponding segment of the input was generated. Results are summarized in Figures 5 and 6.

into 4 segments each (see top of Figure 4), were chosen as template patterns. Other spike trains over 1000 ms were generated by choosing for each 250 ms segment one of the two templates for this segment, and by jittering each spike in the templates (more precisely: each spike was moved by an amount drawn from a Gaussian distribution with mean 0 and a SD that we refer to as "jitter", see bottom of Figure 4). A typical spike train generated in this way is shown in the middle of Figure 4. Because of the noisy dislocation of spikes it was impossible to recognize a specific template from a single interspike interval (and there were no spatial cues contained in this single channel input). Instead, a pattern formed by several interspike intervals had to be recognized and classified retrospectively. Furthermore readouts were not only trained to classify at time $t = 1000$ ms (i.e., at after the input spike train had entered the circuit) the template from which the last 250 ms segment of this input spike train had been generated, but other readouts were trained to classify simultaneously also the templates from which preceding segments of the input (which had entered the circuit several hundred ms earlier) had been generated. Obviously

the latter classification task is substantially more demanding, since the corresponding earlier segments of the input spike train may have left a clear trace in the current firing activity of the recurrent circuit just after they had entered the circuit, but this trace was subsequently overwritten by the next segments of the input spike train (which had no correlation with the choice of the earlier segments). Altogether there were in this experiment 4 readouts f_1 to f_4, where f_i had been trained to classify at time $t = 1000$ ms the i-th independently chosen 250 ms segment of the preceding input spike train.

The performance of the LSM, with a generic recurrent network of 135 I&F neurons as liquid filter (see Appendix B), was evaluated after training of the readout pools on inputs from the same distribution (for jitter = 4 ms), but with an example that the LSM had not seen before. The accuracy of the 4 readouts is plotted in panel A of Figure 5. It demonstrates the fading memory of a generic recurrent circuit of I&F neurons, where information about inputs that occurred several hundred ms ago can be recovered even after that input segment was subsequently overwritten.

Since readout neurons (and neurons within the liquid circuit) were modeled with a realistic time constant of just 30 ms, the question arises where this information about earlier inputs had been stored for

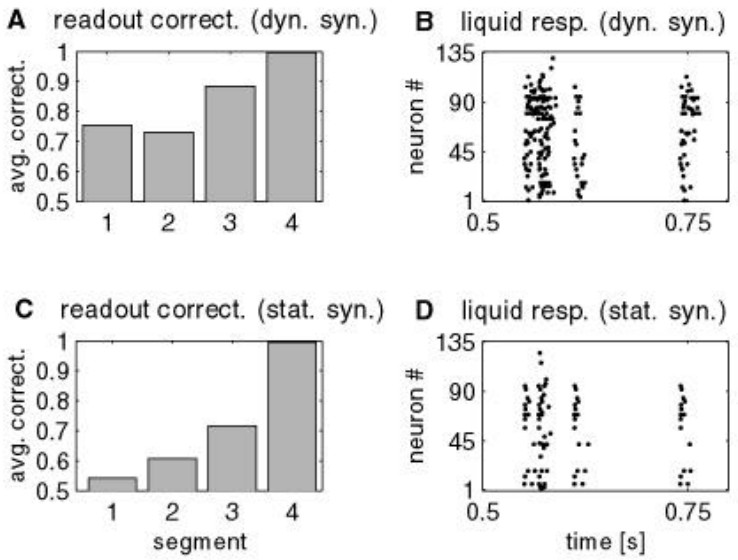

Figure 5: Evaluating the fading memory of a generic neural microcircuit: results. 4 readout modules f_1 to f_4, each consisting of a single perceptron, were trained for their task by linear regression. The readout module f_i was trained to output 1 at time t=1000 ms if the i-th segment of the previously presented input spike train had been constructed from the corresponding template 1, and to output 0 at time t=1000 ms otherwise. Correctness (percentage of correct classification on an independent set of 500 inputs not used for training) is calculated as average over 50 trials. In each trial new Poisson spike trains were drawn as templates, a new randomly connected circuit was constructed (1 column, λ=2; see Appendix B), and the readout modules f_1 to f_4 were trained with1000 training examples generated by the distribution described in Figure 4 . **A**: Average correctness of the 4 readouts for novel test inputs drawn from the same distribution. **B**: Firing activity in the liquid circuit (time interval [0.5 s, 0.8 s]) for a typical input spike train. **C**: Results of a control experiment where all dynamic synapses in the liquid circuit had been replaced by static synapses (the mean values of the synaptic strengths were uniformly re-scaled so that the average liquid activity is approximately the same as for dynamic synapses). The liquid state of this circuit contained substantially less information about earlier input segments. **D**: Firing activity in the liquid circuit with static synapses used for the classification results reported in panel C . The circuit response to each of the 4 input spikes that entered the circuit during the observed time interval [0.5 s, 0.8 s] is quite stereotypical without dynamic synapses (except for the second input spike that arrives just 20 ms after the first one). In contrast the firing response of the liquid circuit with dynamic synapses (panel B) is different for each of the 4 input spikes, showing that dynamic synapses endow these circuits with the capability to process new input differently depending on the context set by preceding input, even if that preceding input occurred several hundred ms before.

several hundred ms. As a control we repeated the same experiment with a liquid circuit where the dynamic synapses had been replaced by static synapses (with synaptic weights that achieved about the same level of firing activity as the circuit with dynamic synapses). Panel C of Fig. 5 shows that this results in a significant loss in performance for the classification of all except for the last input segment. A possible explanation is provided by the raster plots of firing activity in the liquid circuit with (panel B) and without dynamic synapses (panel D), shown here with high temporal resolution. In the circuit with dynamic synapses the recurrent activity differs for each of the 4 spikes that entered the circuit during the time period shown, demonstrating that each new spike is processed by the circuit in an individual manner that depends on the "context" defined by preceding input spikes. In contrast, the firing response is very stereotypical for the same 4 input spikes in the circuit without dynamic synapses, except for the response to the second spike that arrives within 20 ms of the first one (see the period between 500 and 600 ms in panel D). This indicates that the short term dynamics of synapses may play an essential role in the integration of information for real-time processing in neural microcircuits.

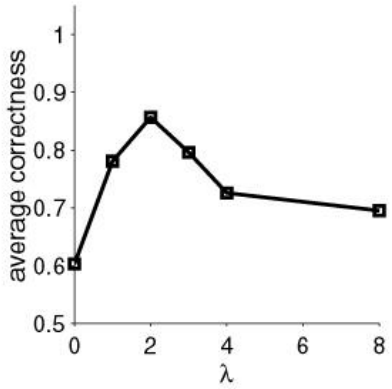

Figure 6: Average correctness depends on the parameter λ that controls the distribution of random connections within the liquid circuit. Plotted is the average correctness (at time t=1000 ms, calculated as average over 50 trials as in Figure 5; same number of training and test examples) of the readout module f_3 (which is trained to classify retroactively the second to last segment of the preceding spike train) as a function of λ. The bad performance for λ=0 (no recurrent connections within the circuit) shows that recurrent connections are essential for achieving a satisfactory separation property in neural microcircuits. Too large values of λ also decrease the performance because they support a chaotic response.

Figure 6 examines another aspect of neural microcircuits that appears to be important for their separation property: the statistical distribution of connection lengths within the recurrent circuit. Six types of liquid circuits, each consisting of 135 I&F neurons but with different values of the parameter λ which regulated the average number of connections and the average spatial length of connections (see Appendix B), were trained and evaluated according to the same protocol and for the same task as in Fig. 5. Shown in Fig. 6 is for each of these 6 types of liquid circuits the average correctness of the readout f_3 on novel inputs, after it had been trained to classify the second to last segment of the input spike train. The performance was fairly low for circuits without recurrent connections (λ = 0). It also was fairly low for recurrent circuits with large values of λ, whose largely length-independent distribution of connections homogenized the microcircuit and facilitated chaotic behavior. Hence for this classification task the ideal "liquid circuit" is a microcircuit that has in addition to local connections to neighboring neurons also a few long-range connections, thereby interpolating between the customarily considered extremes of strictly total connectivity (like in a cellular automaton) on one hand, and the locality-ignoring global connectivity of a Hopfield net on the other hand.

The performance results of neural implementations of LSMs that were reported in this section should not be viewed as absolute data on the computational power of recurrent neural circuits. Rather the general theory suggests that their computational power increases with any improvement in their separation or approximation property. Since the approximation property *AP* was already close to optimal for these networks (increasing the number of neurons in the readout module did not increase the performance significantly; not shown), the primary limitation in performance lay in the separation property *SP*. Intuitively it is clear that the liquid circuit needs to be sufficiently complex to hold the details required for the particular task, but should reduce information that is not relevant to the task (for example spike time jitter). *SP* can be engineered in many ways such as incorporating neuron diversity, implementing specific synaptic architectures, altering microcircuit connectivity, or simply recruiting more columns. The last option is of particular interest because it is not available in most computational models. It will be explored in the next section.

Adding Computational Power

An interesting structural difference between neural systems and our current generation of artificial computing machinery is that the computational power of neural systems can apparently be enlarged by recruiting more circuitry (without the need to rewire old or new circuits). We explored the consequences of recruiting additional columns for neural implementations of LSMs (see panel B of Fig. 7), and compared it with the option of just adding further connections to the

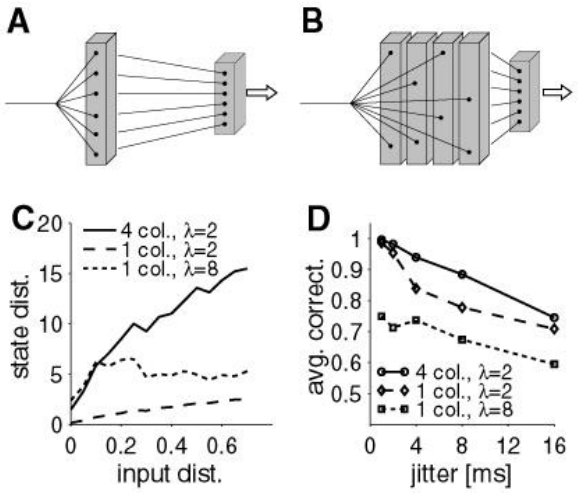

Figure 7: Separation property and performance of liquid circuits with larger numbers of connections or neurons. **A** and **B**: Schematic drawings of LSMs consisting of one column (A) and four columns (B). Each column consists of 3×3×15 = 135 I&F neurons. **C**: Separation property depends on the structure of the liquid. Average state distance (at time $t = 100$ ms) calculated as described in Figure 2. A column with high internal connectivity (high λ) achieves higher separation as a single column with lower connectivity, but tends to chaotic behavior where it becomes equally sensitive to small and large input differences $d(u,v)$. On the other hand the characteristic curve for a liquid consisting of 4 columns with small λ is lower for values of $d(u,v)$ lying in the range of jittered versions u and v of the same spike train pattern ($d(u,v) \leq 0.1$ for jitter $\leq$ 8 ms) and higher for values of $d(u,v)$ in the range typical for spike trains u and v from different classes (mean: 0.22). **D**: Evaluation of the same three types of liquid circuits for noise robust classification. Plotted is the average performance for the same task as in Fig. 6, but for various values of the jitter in input spike times. Several columns (not interconnected) with low internal connectivity yield a better performing implementation of a LSM for this computational task, as predicted by the analysis of their separation property.

primary one-column-liquid that we used so far (135 I&F neurons with $\lambda = 2$, see panel A of Fig. 7). Panel C of Fig. 7 demonstrates that the recruitment of additional columns increases the separation property of the liquid circuit in a desirable manner, where the distance between subsequent liquid states (always recorded at time $t = 1000$ ms in this experiment) is proportional to the distance between the spike train inputs that had previously entered the liquid circuit (spike train distance measured in the same way as for Fig. 2). In contrast the addition of more connections to a single column ($\lambda = 8$, see Appendix B) also increases the separation between subsequent liquid states, but in a quasi-chaotic manner where small input distances cause about the same distances between subsequent liquid states as small input differences. In particular the subsequent liquid state distance is about equally large for two jittered versions of the input spike train state (yielding typically a value of $d(u,v)$ around 0.1) as for significantly different input spike trains that require different outputs of the readouts. Thus improving *SP* by altering the intrinsic microcircuitry of a single column increases sensitivity for the task, but also increases sensitivity to noise. The performance of these different types of liquid circuits for the same classification task as in Fig. 6 is consistent with this analysis of their characteristic separation property. Shown in panel D of Fig. 7 is their performance for various values of the spike time jitter in the input spike trains. The optimization of *SP* for a specific distribution of inputs and a specific group of readout modules is likely to arrive at a specific balance between the intrinsic complexity of the microcircuitry and the number of repeating columns.

Parallel Computing in Real-Time on Novel Inputs

Since the liquid of the LSM does not have to be trained for a particular task, it supports parallel computing in real-time. This was demonstrated by a test in which multiple spike trains were injected into the liquid and multiple readout neurons were trained to perform different tasks in parallel. We added 6 readout modules to a liquid consisting of 2 columns with different values of λ[7]. Each of the 6 readout modules was trained independently for a completely different online task that required an output value at *any time t*. We focused here on tasks that require diverse and rapidly changing analog output responses *y(t)*. Figure 8 shows that after training each of these 6 tasks can be performed in real-time with high accuracy. The performance shown is for a novel input that was not drawn from the same distribution as the training examples, and differs in several aspects from the training examples (thereby demonstrating the possibility of extra-generalization in neural microcircuits, due to their inherent bias, that goes beyond the usual definition of generalization in statistical learning theory).

Readout-Assigned Equivalent States of a Dynamical System

Real-time computation on novel inputs implies that the readout must be able to generate an invariant or appropriately scaled response for any input even though the liquid state may never repeat. Indeed, Figure 3 showed already that the dynamics of readout pools can become quite independent from the dynamics of the liquid even though the liquid neurons are the only source

[7] In order to combine high sensitivity with good generalization performance we chose here a liquid consisting of two columns as before, one with λ=2, the other with λ=8 and the interval [14.0 14.5] for the uniform distribution of the nonspecific background current I_b.

of input. To examine the underlying mechanism for this relatively independent readout response, we re-examined the readout pool from Figure 3. Whereas the firing activity within the liquid circuit was highly dynamic, the firing activity in the readout pool was almost constant after training. The stability of the readout response does not simply come about because the readout only samples a few "unusual" liquid neurons as shown by the distribution of synaptic weights onto a sample readout neuron (Figure 9F). Since the synaptic weights do not change after learning, this indicates that the readout neurons have learned to define a notion of equivalence for dynamic states of the liquid. Indeed, equivalence classes are an inevitable consequence of collapsing the high dimensional space of liquid states into a single dimension, but what is surprising is that the equivalence classes are meaningful in terms of the task, allowing invariant and appropriately scaled readout responses and therefore real-time computation on *novel inputs.* Furthermore, while the input rate may contain salient information that is constant for a particular readout element, it may not be for another (see for example Fig. 8), indicating that equivalence classes and dynamic stability exist purely from the perspective of the readout elements.

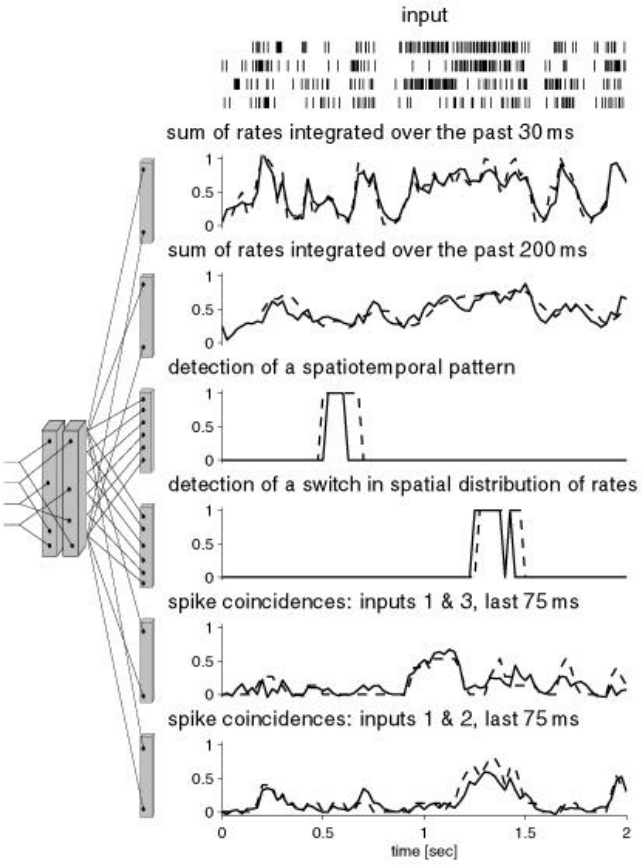

Figure 8: Multi-tasking in real-time. 4 input spike trains of length 2 s (shown at the top) are injected into a liquid module consisting of 2 columns (randomly constructed with the same parameters; see Appendix B), which is connected to multiple readout modules. Each readout module is trained to extract information for a different real-time computing task. The target functions are plotted as dashed line, and population response of the corresponding readout module as solid line. The tasks assigned to the 6 readout modules were the following: Represent the *sum of rates:* at time t, output the sum of firing rates of all 4 input spike trains within the last 30ms. Represent the *integral of the sum of rates:* at time t, output the total activity in all 4 inputs integrated over the last 200ms. *Pattern detection*: output a high value if a specific *spatio temporal spike pattern* appears. Represent a *switch in spatial distribution of rates:* output a high value if a specific input pattern occurs where the rate of input spike trains 1 and 2 goes up and simultaneously the rate of input spike trains 3 and 4 goes down, otherwise remain low. Represent the *firing correlation*: at time t, output the number of spike coincidences (normalized into the range [0 1]) during the last 75 ms for inputs 1 and 3 and separately for inputs 1 and 2. Target readout values are plotted as dashed lines, actual outputs of the readout modules as solid lines, all in the same time scale as the 4 spike trains shown at the top that enter the liquid circuit during this 2 s time interval.

Results shown are for a novel input that was not drawn from the same distribution as the training examples. 150 training examples were drawn randomly from the following distribution. Each input spike train was an independently drawn Possion spike train with a time varying rate of $r(t) = A+B \sin(2 \pi f t + \alpha)$. The parameters A, B, and f where drawn randomly from the following intervals (the phase was fixed at α=0° deg): A [0Hz, 30Hz] and [70Hz, 100Hz], B [0Hz, 30Hz] and [70Hz, 100Hz], f [0.5Hz, 1Hz] and [3Hz, 5Hz]. On this background activity 4 different patterns had been superimposed (always in the same order during training): rate switch to inputs 1 and 3, a burst pattern, rate switch to inputs 1 and 2, and finally a spatio temporal spike pattern.

The results shown are for a test input that could not be generated by the same distribution as the training examples, because its base level (A=50Hz), as well as the amplitude (B=50Hz), frequency (f=2Hz) and phase (α=180° deg) of the underlying time varying firing rate of the Poisson input spike trains were chosen to lie in the middle of the gaps between the two intervals that were used for these parameters during training. Furthermore the spatio-temporal patterns (a burst pattern, rate switch to inputs 1 and 3, and rate switch to inputs 1 and 2), that were superimposed to achieve more input variation within the observed 2 s, never occured in this order and at these time points for any training input. Hence the accurate performance for this novel input demonstrates substantial generalization capabilities of the readouts after training.

Discussion

We introduce the liquid state machine, a new paradigm for real-time computing on time-varying input streams. In contrast to most computational models it does not require the construction of a circuit or program for a specific computational task. Rather, it relies on principles of high-dimensional dynamical systems and learning theory that allow it to adapt unspecific evolved or found recurrent circuitry for a given computational task. Since only the readouts, not the recurrent circuit itself, have to be adapted for specific computational tasks, the same recurrent circuit can support completely different real-time computations in parallel. The underlying abstract computational model of a liquid state machine (LSM) emphasizes the importance of perturbations in dynamical systems for real-time computing, since even without stable states or attractors the separation property and the approximation property may endow a dynamical system with virtually unlimited computational power on time-varying inputs.

In particular we have demonstrated the computational universality of generic recurrent circuits of integrate-and-fire neurons (even with quite arbitrary connection structure), if viewed as special cases of LSMs. Apparently this is the first stable and generally applicable method for using generic recurrent networks of integrate-and-fire neurons to carry out a wide family of complex real-time computations on spike trains as inputs. Hence this approach provides a platform for exploring the computational role of specific aspects of biological neural microcircuits. The computer simulations reported in this article provide possible explanations not only for the computational role of the highly recurrent connectivity structure of neural circuits, but also for their characteristic distribution of connection lengths, which places their connectivity structure between the extremes of strictly local connectivity (cellular automata or coupled map lattices) and uniform global connectivity (Hopfield nets) that are usually addressed in theoretical studies. Furthermore our computer simulations suggest an important computational role of dynamic synapses for real-time computing on time-varying inputs. Finally, we reveal a most unexpected and remarkable principle that readout elements can establish their own equivalence relationships on high-dimensional transient states of a dynamical system, making it possible to generate stable and appropriately scaled output responses even if the internal state never converges to an attractor state.

In contrast to virtually all computational models from computer science or artificial neural networks, this computational model is enhanced rather than hampered by the presence of diverse computational units. Hence it may also provide insight into the computational role of the complexity and diversity of neurons and synapses (see for example (Gupta et al., 2000)).

While there are many plausible models for spatial aspects of neural computation, a biologically realistic framework for modeling temporal aspects of neural computation has been missing. In contrast to models inspired by computer science, the liquid state machine does not try to reduce these temporal aspects to transitions between stable states or limit cycles, and it does not require delay lines or buffers. Instead it proposes that the trajectory of internal states of a recurrent neural circuit provides a raw, unbiased, and universal source of temporally integrated information, from which specific readout elements can extract specific information about past inputs for their individual task. Hence the notorious trial-to-trial stimulus response variations in single and populations of neurons observed experimentally, may reflect an accumulation of information from previous inputs in the trajectory of internal states, rather than noise (see also (Arieli et al., 1996)). This would imply that averaging over trials or binning, peels out most of the information processed by recurrent microcircuits and leaves mostly topographic information.

This approach also offers new ideas for models of the computational organisation of cognition. It suggests that it may not be necessary to scatter all information about sensory input by recoding it through feedforward processing as output vector of an ensemble of feature

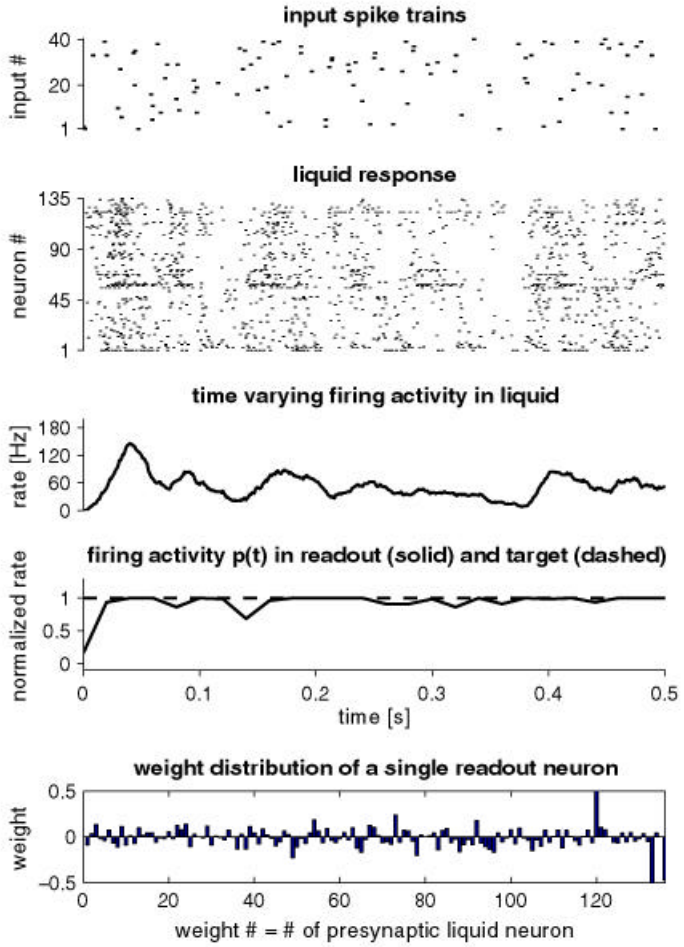

Figure 9: Readout assigned equivalent states of a dynamical system. A LSM (liquid circuit as in Figure 3) was trained for the classification task as described in Figure 3. Results shown are for a novel test input (drawn from the same distribution as the training examples). **A**: The test input consists of 40 Poisson spike trains, each with a constant rate of 5 Hz. **B**: Raster plot of the 135 liquid neurons in response to this input. Note the large variety of liquid states that arise during this time period. **C**: Population rate of the liquid (bin-size 20 ms). Note that this population rate changes quit a bit over time. **D**: Readout response (solid line) and target response (dashed line). The target response had a constant value of 1 for this input. The output of the trained readout module is also almost constant for this test example (except for the beginning), although its input, the liquid states of the recurrent circuit, varied quit a bit during this time period. **F**: Weight distribution of a single readout neuron.

detectors with fixed receptive fields (thereby creating the "binding problem"). It proposes that at the same time more global information about preceding inputs can be preserved in the trajectories of very high dimensional dynamical systems, from which multiple readout modules extract and combine the information needed for their specific tasks. This approach is nevertheless compatible with experimental data that confirm the existence of special maps of feature detectors. These

could reflect specific readouts, but also specialized components of a liquid circuit, that have been optimized genetically and through development to enhance the separation property of a neural microcircuit for a particular input distribution. The new conceptual framework presented in this article suggests to complement the experimental investigation of neural coding by a systematic investigation of the trajectories of internal states of neural microcircuits or systems, which are compared on one hand with inputs to the circuit, and on the other hand with responses of different readout projections.

The liquid computing framework suggests that recurrent neural microcircuits, rather than individual neurons, might be viewed as basic computational units of cortical computation, and therefore may give rise to a new generation of cortical models that link LSM "columns" to form cortical areas where neighboring columns read out different aspects of another column and where each of the stereotypic columns serve both liquid and readout functions. In fact, the classification of neurons into liquid- and readout neurons is primarily made for conceptual reasons. Another conceptual simplification was made by restricting plasticity to synapses onto readout neurons. However synapses in the liquid circuit are likely to be also plastic, for example to support the extraction of independent components of information about preceding time varying inputs for a particular distribution of natural stimuli and thereby enhance the separatio property of neural microcircuits. This plasticity within the liquid would be input-driven and less task specific, and might be most prominent during development of an organism. In addition, the information processing capabilities of hierarchies – or other structured networks – of LSMs remain to be explored, which may provide a basis for modeling larger cortical areas.

Apart from biological modeling, the computational model discussed in this article may also be interest for some areas of computer science. In computer applications where real-time processing of complex input streams is required, such as for example in robotics, there is no need to work with complicated heterogeneous recurrent networks of integrate-and-fire neurons as in biological modeling. Instead, one can use simple devices such as tapped delay lines for storing information about past inputs. Furthermore one can use any one of large selection of powerful tools for static pattern recognition (such as feedforward neural networks, support vector machines, or decision trees) to extract from the current content of such tapped delay line information about a preceding input time series, in order to predict that time series, to classify that time series, or to propose actions based on that time series. This works fine, except that one has deal with the problems caused by local minima in the error functions of such highly nonlinear pattern recognition devices, which may result in slow learning and suboptimal generalization. In general the escape from such local minima requires further training examples, or time-consuming offline computations such as repetition of backprop for many different initial weights, or the solution of a quadratic optimization problem in the case of support vector machines. Hence these approaches tend to be incompatible with real-time requirements, where a classification or prediction of the past input time series is instantly needed. Furthermore these standard approaches provide no support for multi-tasking, since one has to run for each individual classification or prediction task a separate copy of the time-consuming pattern recognition algorithm. In contrast, the alternative computational paradigm discussed in this article suggests to replace the tapped delay line by a nonlinear online projection of the input time series into a high-dimensional space, in combination with linear readouts from that high-dimensional intermediate space. The nonlinear online preprocessing could even be implemented by inexpensive (even partially faulty) analog circuitry, since the details of this online preprocessing do not matter, as long as the separation property is satisfied for all relevant inputs. If this task-independent online

preprocessing maps input streams into a sufficiently high-dimensional space, all subsequent linear pattern recognition devices, such as perceptrons, receive essentially the same classification and regression capability for the time varying inputs to the system as nonlinear classifiers without preprocessing. The training of such linear readouts has an important advantage compared with training nonlinear readouts. While the error minimization for a nonlinear readout is likely to get stuck in local minima, the sum of squared errors for a linear readout has just a single local minimum, which is automatically the global minimum of this error function. Furthermore the weights of a linear readouts can be adapted in an online manner by very simple local learning rules so that the weight vector moves towards this global minimum. Related mathematical facts are exploited by support vector machines in machine learning (Vapnik, 1998), although the boosting of the expressive power of linear readouts is implemented there in a different fashion that is not suitable for real-time computing.

Finally, the new approach towards real-time neural computation presented in this article may provide new ideas for neuromorphic engineering and analog VLSI. Besides implementing recurrent circuits of spiking neurons in silicon one could examine a wide variety of other materials and circuits that may potentially enable inexpensive implementation of liquid modules with suitable separation properties, to which a variety of simple adaptive readout devices may be attached to execute multiple tasks.

Acknowledgement

We would like to thank Rodney Douglas, Herbert Jaeger, Wulfram Gerstner, Alan Murray, Misha Tsodyks, Thomas Poggio, Lee Segal, Tali Tishby, Idan Segev, Phil Goodman & Mark Pinsky for their comments on a draft of this article. The work was supported by project # P15386 of the Austrian Science Fund, the NeuroCOLT project of the EU, the Office of Naval Research, HFSP, Dolfi & Ebner Center and the Edith Blum Foundation. HM is the incumbent of the Diller Family Chair in Neuroscience.

References

Arieli, A., Sterkin, A., Grinvald, A., & Aertsen, A. (1996). Dynamics of ongoing activity: explanation of the large variability in evoked cortical responses. *Science*, 273, 1868-1871.

Auer, P., Burgsteiner, H., & Maass, W. (2001) *The p-delta rule for parallel perceptrons*. Submitted for publication, online available at http://www.igi.TUGraz.at/maass/p_delta_learning.pdf.

Boyd, S., & Chua, L.O. (1985). Fading memory and the problem of approximating nonlinear operators with Volterra series. *IEEE Trans. on Circuits and Systems*, 32, 1150-1161.

Buonomano, D.V., & Merzenich, M.M. (1995). Temporal information transformed into spatial code by a neural network with realistic properties. Science, 267, 1028-1030.

Dominey P., Arbib, M., & Joseph, J.P. (1995). A model of corticostriatal plasticity for learning oculomotor association and sequences. *J. Cogn. Neurosci.* 7(3), 311-336.

Douglas, R., & Martin, K. (1998). Neocortex. In: *The Synaptic Organization of the Brain*. G.M. Shepherd, Ed. (Oxford University Press), 459-509.

Giles, C.L., Miller, C.B., Chen, D., H.H., Sun, G.Z., & Lee, Y.C. (1992). Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4, 393-405.

Gupta, A., Wang, Y., & Markram, H. (2000). Organizing principles for a diversity of GABAergic interneurons and synapses in the neocortex. *Science* 287, 2000, 273-278.

Hertz, J., Krogh, A., & Palmer, R.G. (1991). *Introduction to the Theory of Neural Computation*. (Addison-Wesley, Redwood City, Ca).

Holden, A.V., Tucker, J.V., & Thompson, B.C. (1991). Can excitable media be considered as computational systems? *Physica D*, 49, 240-246.

Hopfield, J.J., & Brody, C.D. (2001). What is a moment? Transient synchrony as a collective mechanism for spatio-temporal integration. *Proc. Natl. Acad. Sci.*, USA, 89(3), 1282.

Hyoetyniemi, H. (1996). Turing machines are recurrent neural networks. *Proc. of SteP'96 – Genes, Nets and Symbols*. Alander, J., Honkela, T. & Jacobsson, M., editors, Finnish Artificial Intelligence Society, 13-24

Jaeger, H. (2001). The "echo state" approach to analyzing and training recurrent neural networks, submitted for publication.

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

Maass, W. (2000). On the computational power of winner-take-all. *Neural Computation*, 12(11):2519-2536.

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

Maass, W., & Sontag, E.D. (2000). Neural systems as nonlinear filters. *Neural Computation*, 12(8):1743-1772

Markram, H., Wang, Y., & Tsodyks, M. (1998). Differential signaling via the same axon of neocortical pyramidal neurons. *Proc. Natl. Acad. Sci.*, 95, 5323-5328.

Moore, C. (1998). Dynamical recognizers: real-time language recognition by analog computers. *Theoretical Computer Science*, 201, 99-136.

Pearlmutter, B.A. (1995). Gradient calculation for dynamic recurrent neural networks: a survey. *IEEE Trans. On Neural Networks*, 6(5): 1212-1228

Pollack, J.B. (1991). The induction of dynamical recognizers. *Machine Learning*, 7, 227-252.

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

Shepherd, G.M. (1988). A basic circuit for cortical organization. In: *Perspectives in Memory Research*, M. Gazzaniga, Ed. (MIT Press), 93-134.

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

Tsodyks, M., Uziel, A., & Markram, H. (2000). Synchrony generation in recurrent networks with frequency-dependent synapses. *J. Neuroscience*, Vol. 20 RC50.

Vapnik, V.N. (1998). *Statistical Learning Theory*. John Wiley, New York.

von Melchner, L., Pallas, S.L., & Sur, M. (2000). Visual behaviour mediated by retinal projection directed to the auditory pathway. *Nature*, 2000 Apr 20; 404:871-6.

Appendix A: Mathematical Theory

We say that a class CB of filters has the *point-wise separation property* with regard to input functions from U^n if for any two functions $u(\cdot), v(\cdot) \in U^n$ with $u(s) \neq v(s)$ for some $s \leq 0$ there exists some filter $B \in CB$ that separates $u(\cdot)$ and $v(\cdot)$, i.e., $(Bu)(0) \neq (Bv)(0)$. Note that it is *not* required that there exists a filter $B \in CB$ with $(Bu)(0) \neq (Bv)(0)$ for any two functions $u(\cdot), v(\cdot) \in U^n$ with $u(s) \neq v(s)$ for some $s \leq 0$. Simple examples for classes CB of filters that have this property are the class of all delay filters $u(\cdot) \mapsto u^{t_0}(\cdot)$ (for $t_0 \in \mathbf{R}$), the class of all linear filters with impulse responses of the form $h(t) = e^{-at}$ with $a > 0$, and the class of filters defined by standard models for dynamic synapses, see (Maass and Sontag, 2000). A liquid filter L^M of a LSM M is said to be *composed* of filters from CB if there are finitely many filters $B_1, \ldots, B_m$ in CB – to which we refer as basis filters in this context – so that $(L^M u)(t) = \langle (B_1 u)(t), \ldots, (B_m u)(t) \rangle$ for all $t \in \mathbf{R}$ and all input functions $u(\cdot)$ in U^n. In other words: the output of L^M for a particular input *u* is simply the vector of outputs given by these finitely many basis filters for this input *u*.

A class CF of functions has the *approximation property* if for any $m \in \mathrm{N}$, any compact (i.e., closed and bounded) set $X \subseteq \mathbf{R}^m$, any continuous function $h: X \to \mathbf{R}$ and any given $\boldsymbol{r} > 0$ there exists some f in CF so that that $|h(x) - f(x)| \leq \boldsymbol{r}$ for all $x \in X$. The definition for the case of functions with multi-dimensional output is analogous.

Theorem 1: Consider a space U^n of input functions where $U = \{u : \mathbf{R} \to [-B, B] : |u(t) - u(s)| \leq B' \cdot |t - s| \text{ for all } t, s \in \mathbf{R}\}$ for some $B, B' > 0$ (thus U is a class of uniformly bounded and Lipschitz-continuous functions). Assume that CB is some arbitrary class of time invariant filters with fading memory that has the point-wise separation property. Furthermore, assume that CF is some arbitrary class of functions that satisfies the approximation property. Then any given time invariant filter F that has fading memory can be approximated by LSMs with liquid filters L^M composed from basis filters in CB and readout maps f^M chosen from CF. More precisely: For every $\boldsymbol{e} > 0$ there exist $m \in \mathrm{N}$, $B_1, \ldots, B_m \in CB$ and $f^M \in CF$ so that the output $y(\cdot)$ of the liquid state machine M with liquid filter L^M composed of $B_1, \ldots, B_m$, i.e., $(L^M u)(t) = \langle (B_1 u)(t), \ldots, (B_m u)(t) \rangle$, and readout map f^M satisfies for all $u(\cdot) \in U^n$ and all $t \in \mathbf{R}$ $\| (Fu)(t) - y(t) \| \leq \boldsymbol{e}$.

The *proof* of this theorem follows from the Stone-Weierstrass Approximation Theorem, similarly as the proof of Theorem 1 in Boyd & Chua (1985).One can easily show that the inverse of Theorem 1 also holds: If the functions in CF are continuous, then any filter F that can be approximated by the liquid state machines considered in Theorem 1 is time invariant and has fading memory. In combination with Theorem 1, this provides a complete characterization of the computational power of LSMs.

In order to extend Theorem 1 to the case where the inputs are finite or infinite spike trains, rather than continuous functions of time, one needs to consider an appropriate notion of fading memory for filters on spike trains. The traditional definition, given in footnote 4, is not suitable for the following reason. If $u(\cdot)$ and $v(\cdot)$ are functions with values in {0, 1} that represent spike trains and if $\delta \leq 1$, then the condition $\|u(t) - v(t)\| < \delta$ is too strong: it would require that $u(t) = v(t)$. Hence we define for the case where the domain U consists of spike trains (i. e. $0-1$ valued functions) that a filter $F : U^n \to (\mathbf{R}^{\mathbf{R}})^k$ has *fading memory on spike trains* if for every $u = \langle u_1, ..., u_n \rangle \in U^n$ and every $\varepsilon > 0$ there exist $\delta > 0$ and $m \in \mathbb{N}$ so that $\|(Fv)(0) - (Fu)(0)\| < \varepsilon$ for all $v = \langle v_1, ..., v_n \rangle \in U^n$ with the property that for $i = 1,...,n$ the last m spikes in v_i (before time 0) each have a distance of at most δ from the corresponding ones among the last m spikes in u_i. Intuitively this says that a filter has fading memory on spike trains if the most significant bits of the filter output can already be determined from the approximate times of the last few spikes in the input spike train. For this notion of fading memory on spike trains one can prove:

Theorem 2: Consider the space U^n of input functions where U is the class of spike trains with some minimal distance Δ between successive spikes (e. g., $\Delta = 1$ ms). Assume that CB is some arbitrary class of time invariant filters with fading memory on spike trains that has the point-wise separation property. Furthermore, assume that CF is some arbitrary class of functions that satisfies the approximation property. Then any given time invariant filter F that has fading memory on spike trains can be approximated by liquid state machines with liquid filters L^M composed from basis filters in CB and readout maps f^M chosen from CF.

The *proof* for Theorem 2 is obtained by showing that for all filters F fading memory on spike trains is equivalent to continuity with regard to a suitable metric on spike trains that turns the domain of spike trains into a *compact* metric space. Hence, one can apply the Stone-Weierstrass Approximation Theorem also to this case of computations on spike trains.

Appendix B: Details of the Computer Simulation

We used a randomly connected circuit consisting of 135 integrate-and-fire neurons, 20% of which were randomly chosen to be inhibitory, as a single "column" of neural circuitry (Tsodyks et al., 2000). Neuron parameters: membrane time constant 30ms, absolute refractory period 3ms (excitatory neurons), 2ms (inhibitory neurons), threshold 15mV (for a resting membrane potential assumed to be 0), reset voltage 13.5mV, constant nonspecific background current I_b =13.5nA, input resistance 1 MΩ.

Connectivity structure: The probability of a synaptic connection from neuron a to neuron b (as well as that of a synaptic connection from neuron b to neuron a) was defined as $C \cdot e^{-(D(a,b)/\lambda)^2}$, where λ is a parameter which controls both the average number of connections and the average distance between neurons that are synaptically connected. We assumed that the 135 neurons were located on the integer points of a 15×3×3 column in space, where $D(a,b)$ is

the Euclidean distance between neurons a and b. Depending on whether a and b were excitatory (E) or inhibitory (I), the value of C was 0.3 (*EE*), 0.2 (*EI*), 0.4 (*IE*), 0.1 (*II*).

In the case of a synaptic connection from a to b we modeled the synaptic dynamics according to the model proposed in (Markram et al., 1998), with the synaptic parameters U (use), D (time constant for depression), F (time constant for facilitation) randomly chosen from Gaussian distributions that were based on empirically found data for such connections. Depending on whether a, b were excitatory (E) or inhibitory (I), the mean values of these three parameters (with D, F expressed in second, s) were chosen to be .5, 1.1, .05 (*EE*), .05, .125, 1.2 (*EI*), .25, .7, .02 (*IE*), .32, .144, .06 (*II*). The SD of each parameter was chosen to be 50% of its mean (with negative values replaced by values chosen from an appropriate uniform distribution). The mean of the scaling parameter A (in nA) was chosen to be 30 (EE), 60 (EI), -19 (IE), -19 (II). In the case of input synapses the parameter A had a value of 18 nA if projecting onto a excitatory neuron and 9.0 nA if projecting onto an inhibitory neuron.). The SD of the A parameter was chosen to be 100% of its mean and was drawn from a gamma distribution. The postsynaptic current was modeled as an exponential decay $\exp(-t/\tau_s)$ with τ_s=3ms (τ_s=6ms) for excitatory (inhibitory) synapses. The transmission delays between liquid neurons were chosen uniformly to be 1.5 ms (EE), and 0.8 for the other connections. For each simulation, the initial conditions of each leaky-integrate and fire neuron, i.e. the membrane voltage at time t=0, were drawn randomly (uniform distribution) from the interval [13.5mV, 15.0mV]. Together with the spike time jitter in the input these randomly drawn initial conditions served as implementation of noise in our simulations (in order to test the noise robustness of our approach).

Readout elements used in the simulations of Figures 3, 8, and 9 were made of 51 integrate-and-fire neurons (unconnected). A variation of the perceptron learning rule (the delta rule, see (Hertz et al., 1991)) was applied to scale the synapses of these readout neurons: the p-delta rule discussed in (Auer et al., 2001). The p-delta rule is a generalization of the delta rule that trains a population of perceptrons to adopt a given population response (in terms of the number of perceptrons that are above threshold), requiring very little overhead communication. This rule, which formally requires to adjust the weights and the threshold of perceptrons, was applied in such a manner that the background current of an integrate-and-fire neuron is adjusted instead of the threshold of a perceptron (while the firing threshold was kept constant at 15mV). In Fig. 8 and 9 the readout neurons are not fully modeled as integrate and fire neurons, but just as perceptrons (with a low pass filter in front that transforms synaptic currents into PSPs, time constant 30 ms); in order to save computation time. In this case the "membrane potential" of each perceptron is checked every 20 ms, and it is said to "fire" at this time point if this "membrane potential" is currently above the 15 mV threshold. No refractory effects are modeled, and no reset after firing. The percentage of readout neurons that fire during a 20 ms time bin is interpreted as the current output of this readout module (assuming values in [0 , 1]).

In the simulations for Figures 5, 6, and 7 we used just single perceptrons as readout elements. The weights of such a single perceptron have been trained using standard linear regression: the target value for the linear regression problem was +1 (-1) if the perceptron should output 1 (0) for the given input. The output of the perceptron after learning was 1 (0) if the weighted sum of inputs was ≥ 0 (< 0).