Liquid Computing

Wolfgang Maass*

Institute for Theoretical Computer Science
Technische Universitaet Graz
A-8010 Graz, Austria
maass@igi.tugraz.at
http://www.igi.tugraz.at/

Abstract. This review addresses structural differences between that type of computation on which computability theory and computational complexity theory have focused so far, and those computations that are usually carried out in biological organisms (either in the brain, or in the form of gene regulation within a single cell). These differences concern the role of time, the way in which the input is presented, the way in which an algorithm is implemented, and in the end also the definition of what a computation is. This article describes liquid computing as a new framework for analyzing those types of computations that are usually carried out in biological organisms.

1 Introduction

The computation of a Turing machine always begins in a designated initial state q_0, with the input x (some finite string of symbols from some finite alphabet) written on some designated tape. The computation runs until a halt-state is entered (the inscription y of some designated tape segment is then interpreted as the result of the computation). This is a typical example for an *offline computation*, where the complete input x is available at the beginning of the computation, and no trace of this computation, or of its result y, is left when the same Turing machine subsequently carries out another computation for another input $\tilde{x}$ (starting again in state q_0). In contrast, the result of a typical computation in the neuronal system of a biological organism, say the decision about the location y on the ground where the left foot is going to be placed at the next step (while walking or running), depends on several pieces of information: on information from the visual system, from the vestibular system which supports balance control, from the muscles (proprioceptive feedback about their current state), from short term memory (how well did the previous foot placement work?), from long term memory (how slippery is this path at the current weather condition?), from brain systems that have previously decided where to go and at what speed, and on information from various other parts of the neural system. In general these

* Partially supported the Austrian Science Fund FWF, project P17229 and project S9102-N13, and by the European Union, project FP6-015879 (FACETS).

diverse pieces of information arrive at different points in time, and the computation of y has to start before the last one has come in. Furthermore, new information (e.g. visual information and proprioceptive feedback) arrives continuously, and it is left up to the computational system how much of it can be integrated into the computation of the position y of the next placement of the left foot (obviously those organisms have a better chance to survive which also can integrate later arriving information into the computation). Once the computation of y is completed, the computation of the location y' where the right foot is subsequently placed is not a separate computation, that starts again in some neutral initial state q_0. Rather, it is likely to build on pieces of inputs and results of subcomputations that had already been used for the preceding computation of y.

The previously sketched computational task is a typical example for an *online computation* (where input pieces arrive all the time, not in one batch). Furthermore it is an example for a *real-time computation*, where one has a strict deadline by which the computation of the output y has to be completed (otherwise a 2-legged animal would fall). In fact, in some critical situations (e. g. when a 2-legged animal stumbles, or hits an unexpected obstacle) a biological organism is forced to apply an *anytime algorithm*, which tries to make optimal use of intermediate results of computational processing that has occurred up to some externally given time point t_0 (such forced halt of the computation could occur at "any time"). Difficulties in the control of walking for 2-legged robots have taught us how difficult it is to design algorithms which can carry out this seemingly simple computational task. In fact, this computational problem is largely unsolved, and humanoid robots can only operate within environments for which they have been provided with an accurate model. This is perhaps surprising, since on the other hand current computers can beat human champions in seemingly more demanding computational tasks, such as winning a game of chess. One might argue that one reason, why walking in a new terrain is currently a computationally less solved task, is that computation theory and algorithm design have focused for several decades on offline computations, and have neglected seemingly mundane computational tasks such as walking. This bias is understandable, because evolution had much more time to develop a computational machinery for the control of human walking, and this computational machinery works so well that we don't even notice anymore how difficult this computational task is.

2 Liquid State Machines

A computation machine M that carries out online computations typically computes a function F that does not map input numbers or (finite) bit strings onto output numbers or bit strings, but input streams onto output streams. These input- and output streams are usually encoded as functions $u : \mathbb{Z} \to \mathbb{R}^n$ or $u : \mathbb{R} \to \mathbb{R}^n$, where the argument t of $u(t)$ is interpreted as the (discrete or continuous) time point t when the information that is encoded by $u(t) \in \mathbb{R}^n$

becomes available. Hence such computational machine M computes a function of higher type (usually referred to as operator, functional, or filter), that maps input functions u from some domain U onto output functions y. In lack of a better term we will use the term filter in this article, although filters are often associated with somewhat trivial signal processing or preprossessing devices. However, one should not fall into the trap of identifying the general term of a filter with special classes of filters such as linear filters. Rather one should keep in mind that any input to any organism is a function of time, and any motor output of an organism is a function of time. Hence biological organisms compute filters. The same holds true for any artificial behaving system, such as a robot.

We will only consider computational operations on functions of time that are input-driven, in the sense that the output does not depend on any absolute internal clock of the computational device. Filters that have this property are called time invariant. Formally one says that a filter F is *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 by the same amount t_0, i.e., $(Fu^{t_0})(t) = (Fu)(t+t_0)$ for all $t, t_0 \in \mathbb{R}$, where u^{t_0} is the function defined by $u^{t_0}(t) := u(t + t_0)$. Note that if the domain U of input functions $u(\cdot)$ is closed under temporal shifts, then a time invariant filter $F : U \to \mathbb{R}^{\mathbb{R}}$ is identified uniquely by the values $y(0) = (Fu)(0)$ of its output functions $y(\cdot)$ at time 0. In other words: in order to identify or characterize a time invariant filter F we just have to observe its output values at time 0, while its input varies over all functions $u(\cdot) \in U$. Hence one can replace in the mathematical analysis such filter F by a functional, i.e. a simpler mathematical object that maps input functions onto real values (rather than onto functions of time).

Various theoretical models for analog computing are of little practical use because they rely on hair-trigger decisions, for example they allow that the output is 1 if the value of some real-valued input variable u is ≥ 0, and 0 otherwise. Another unrealistic aspect of some models for computation on functions of time is that they automatically allow that the output of the computation depends on the full infinitely long history of the input function $u(\cdot)$. Most practically relevant models for analog computation on continuous input streams degrade gracefully under the influence of noise, i.e. they have a fading memory. *Fading memory* is a continuity property of filters F, which requires that for any input function $u(\cdot) \in U$ the output $(Fu)(0)$ can be approximated by the outputs $(Fv)(0)$ for any other input functions $v(\cdot) \in U$ that approximate $u(\cdot)$ on a sufficiently long time interval $[-T, 0]$ in the past. Formally one defines that $F : U \to \mathbb{R}^{\mathbb{R}}$ 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 $v \in U$ 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 $u(\cdot)$ in some finite time interval $[-T, 0]$. 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 have knowledge about values of $u(\cdot)$ for more than a finite time

interval back into the past.

The universe of time-invariant fading memory filters is quite large. It contains all filters F that can be characterized by Volterra series, i.e. all filters F whose output $(Fu)(t)$ is given by a finite or infinite sum (with $d = 0, 1, \ldots$) of terms of the form $\int_0^\infty \ldots \int_0^\infty h_d(\tau_1, \ldots, \tau_d) \cdot u(t-\tau_1) \cdot \ldots \cdot u(t-\tau_d) d\tau_1 \ldots d\tau_d$, where some integral kernel h_d is applied to products of degree d of the input stream $u(\cdot)$ at various time points $t - \tau_i$ back in the past. In fact, under some mild conditions on the domain U of input streams the class of time invariant fading memory filters coincides with the class of filters that can be characterized by Volterra series.

In spite of their complexity, all these filters can be uniformly approximated by the simple computational models M of the type shown in Fig. 1, which had been introduced in [1]:

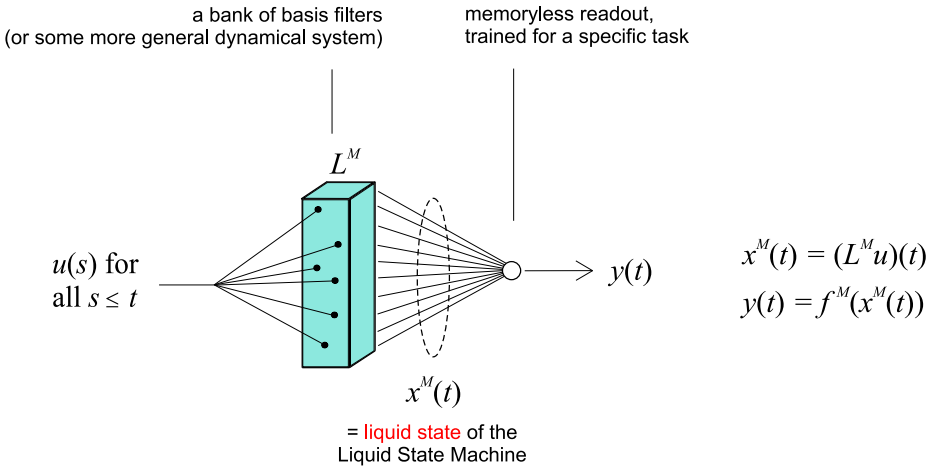


Fig. 1. Structure of a Liquid State Machine (LSM) M, which transforms input streams $u(\cdot)$ into output streams $y(\cdot)$. If L^M is a bank of k basis filters, the "liquid state" $\mathbf{x}^M(t) \in \mathbb{R}^k$ is the output of those k filters at time t. If L^M is a more general dynamical system, $\mathbf{x}^M(t)$ is that part of its current internal state that is "visible" for a readout.

Theorem 1. (based on [2]; see Theorem 3.1 in [3] for a detailed proof). *Any filter F defined by a Volterra series can be approximated with any desired degree of precision by the simple computational model M shown in Fig. 1*

– *if there is a rich enough pool* $\mathbf{B}$ *of basis filters (time invariant, with fading memory) from which the basis filters $B_1, \ldots, B_k$ in the filterbank L^M can be*

*chosen (***B** *needs to have the pointwise separation property) and*
- *if there is a rich enough pool* **R** *from which the readout functions f can be chosen (***R** *needs to have the universal approximation property, i.e. any continuous function on a compact domain can be uniformly approximated by functions from* **R***).*

Definition: *A class* **B** *of basis filters has the pointwise separation property if there exists for any two input functions $u(\cdot), v(\cdot)$ with $u(s) \neq v(s)$ for some $s \leq t$ a basis filter $B \in \mathbf{B}$ with $(Bu)(t) \neq (Bv)(t)$.*

It turns out that many real-world dynamical systems (even a pool of water) satisfy (for some domain U of input streams) at least some weak version of the pointwise separation property, where the outputs $\mathbf{x}^M(t)$ of the basis filters are replaced by some "visible" components of the state vector of the dynamical system. In fact, many real-world dynamical systems also satisfy approximately an interesting kernel property[2], which makes it practically sufficient to use just a *linear* readout function f^M. This is particularly important if L^M is kept fixed, and only the readout f^M is selected (or trained) in order to approximate some particular Volterra series F. Reducing the adaptive part of M to the *linear* readout function f^M has the unique advantage that a learning algorithm that uses gradient descent to minimize the approximation error of M cannot get stuck in local minima of the mean-squared error. The resulting computational model can be viewed as a generalization of a finite state machine to continuous time and continuous ("liquid") internal states $\mathbf{x}^M(t)$. Hence it is called a Liquid State Machine (LSM)[3].

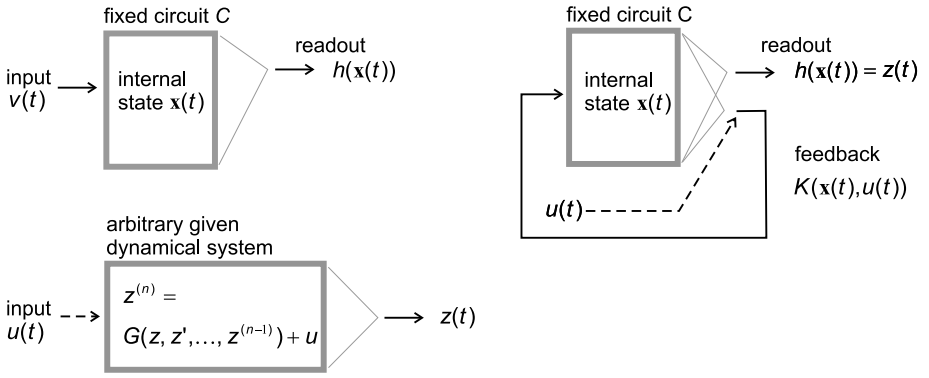
If the dynamical systems L^M have fading memory, then only filters with fading memory can be represented by the resulting LSM's. Hence they cannot approximate arbitrary finite state machines (not even for the case of discrete time and a finite range of values $u(t)$). It turns out that a large jump in computational power occurs if one augments the computational model from Fig. 1 by a feedback from a readout back into the circuit (assume it enters the circuit like an input variable).

Theorem 2. [5]. *There exists a large class S_n of dynamical systems C with fading memory (described by systems of n first order differential equations) that*

[2] A kernel (in the sense of machine learning) is a nonlinear projection Q of n input variables $u_1, \ldots, u_n$ into some high-dimensional space. For example all products $u_i \cdot u_j$ could be added as further components to the n-dimensional input vector $< u_1, \ldots, u_n >$. Such nonlinear projection Q boosts the power of any *linear* readout f applied to $Q(\mathbf{u})$. For example in the case where $Q(\mathbf{u})$ contains all products $u_i \cdot u_j$, a subsequent linear readout has the same expressive capability as quadratic readouts f applied to the original input variables $u_1, \ldots, u_n$. More abstractly, Q should map all inputs $\mathbf{u}$ that need to be separated by a readout onto a set of linearly independent vectors $Q(\mathbf{u})$.

[3] Herbert Jaeger (see [4]) had simultaneously and independently introduced a very similar computational model under the name Echo State Network.

acquire through feedback universal computational capabilities for analog computing. More precisely: through a proper choice of a (memoryless) feedback function K and readout h they can simulate any given dynamical system of the form $z^{(n)} = G(z, z', \ldots, z^{(n-1)}) + u$ with a sufficiently smooth function G:



This holds in particular for neural circuits C defined by differential equations of the form $x_i'(t) = -\lambda_i x_i(t) + \sigma(\sum_{j=1}^{n} a_{ij} x_j(t)) + b_i \cdot \sigma(v(t))$ (under some conditions on the λ_i, a_{ij}, b_i).

If one allows several feedbacks K, such dynamical systems C become universal for n^{th} order dynamical systems defined by a system consisting of a corresponding number of differential equations. Since such systems of differential equations can simulate arbitrary Turing machines [6], these dynamical systems C with a finite number of feedbacks become (according to the Church-Turing thesis) also universal for *digital computation*.

Theorem 2 suggests that even quite simple neural circuits with feedback have in principle unlimited computational power[4]. This suggests that the main problem of a biological organism becomes the *selection* (or learning) of suitable feedback functions K and readout functions h. For dynamical systems C that have a good kernel-property, already *linear* feedbacks and readouts endow such dynamical systems with the capability to emulate a fairly large range of other dynamical systems (or "analog computers").

3 Applications

LSMs had been introduced in [1] (building on preceding work in [7]) in the process of searching for computational models that can help us to understand the

[4] Of course, in the presence of noise this computational power is reduced to that of a finite state machine, see [5] for details.

computations that are carried out in a "cortical microcircuit" [8], i.e. in a local circuit of neurons in the neocortex (say in a "cortical column"). This approach has turned out to be quite successful, since it made it possible to carry out quite demanding computations with circuits consisting of reasonably realistic models for biological neurons ("spiking neurons") and biological synapses ("dynamical synapses"). Note that in this model a large number of different readout neurons can learn to extract different information from the same circuit. One concrete benchmark task that has been considered was the classification ("recognition") of spoken digits [9]. It turned out that already a LSM where the "liquid" consisted of a randomly connected circuit of just 135 spiking neurons performed quite well. In fact, it provided a nice example for "anytime computations", since the linear readout could be trained effectively to guess at "any time", while a digit was spoken, the proper classification of the digit [1, 10]. More recently it has been shown that with a suitable transformation of spoken digits into spike trains one can achieve with this simple method the performance level of state-of-the-art algorithms for speech recognition [11].

The same computational task had also been considered in an amusing and inspiring study of the computational capability of LSMs where a bucket of water was used as the "liquid" L^M (into which input streams were injected via 8 motors), and video-images of the surface of the water were used as "liquid states" $\mathbf{x}^M(t)$ [12]. Also this realization of a LSM was able to carry out speech recognition, but "fortunately" its performance was below that of a LSM with a simulated circuit of neurons as "liquid". This experiment raises two questions:

i) Can interesting new artificial computing devises be designed on the basis of the LSM-paradigm?

ii) Can the LSM-paradigm be used to gain an understanding of the computational role of specific (genetically encoded) details of the components and connectivity structure of circuits of neurons in the neocortex?

Research in the direction i) is currently carried out in the context of optical computing. First results regarding question ii) were recently published in [13].

Perhaps the most exciting research on the LSM-approach is currently carried out in experimental neuroscience. The LSM-approach makes specific experimentally testable predictions regarding the organization of computations in cortical circuits:

a) information from subsequent stimuli is superimposed in the "liquid state" of cortical circuits, but can be recovered by simple linear readouts (see the "separation property" in Theorem 1).

b) cortical circuits produce nonlinear combinations of different input components (kernel property)

c) the activity of different neurons within a cortical column in response to natural stimuli shows a large diversity of individual responses (rather than

evidence that all neurons within a column carry out a specific common computational operation), since a "liquid" can support many different readouts.

A rather clear confirmation of predictions a) and b) has recently been produced at the Max-Plank Institute for Brain Research in Frankfurt [14]. Some earlier experimental studies had provided already evidence for prediction c), but this prediction needs to be tested more rigorously for natural stimuli.

The exploration of potential engineering applications of the computational paradigm discussed in this article were usually carried out with the closely related echo state networks (ESNs) [4], where one uses simpler non-spiking models for neurons in the "liquid", and works with high numerical precision in the simulation of the "liquid" and the training of linear readouts (which makes a lot of sense, since artificial circuits are subject to substantial lower amounts of noise in comparison with biological circuits of neurons). Research in recent years has produced quite encouraging results regarding applications of ESNs to problems in tele-communication [4], robotics [15], reinforcement learning [16], natural language understanding [17], as well as music-production and -perception [18].

4 Discussion

We have argued in this article that Turing machines are not well-suited for modeling computations in biological neural circuits, and proposed liquid state machines (LSMs) as a more adequate modeling framework. They are designed to model real-time computations (as well as anytime computations) on continuous input streams. In fact, it is quite realistic that a LSM can be trained to carry out the online computation task that we had discussed in section 1 (see [19] for a first application to motor control). A characteristic feature of practical implementations of the LSM model is that its "program" consists of the weights $\mathbf{w}$ of a linear readout function. Since these weights can be chosen to be time invariant, they provide suitable targets for learning (while all other parameters of the LSM can be fixed in advance, based on the expected complexity and precision requirement of the computational tasks that are to be learnt). It makes a lot of sense (from the perspective of statistical learning theory) to restrict learning to such weights $\mathbf{w}$, since they have the unique advantage that gradient descent with regard to some mean-square error function $E(\mathbf{w})$ cannot get stuck in local minima of this error function (since $\nabla_{\mathbf{w}} E(\mathbf{w}) = \mathbf{0}$ defines an affine – hence connected – subspace of the weight space).

One can view these weights $\mathbf{w}$ of the linear readout of a LSM as an analogon to the code $< M >$ of a Turing machine M that is simulated by a universal Turing machine. This analogy makes the learning advantage of LSMs clear, since there is no efficient learning algorithm known which allows us to learn the program $< M >$ for a Turing machine M from examples for correct input/output pairs of M. However the examples discussed in section 3 show that an LSM can be trained quite efficiently to approximate a particular map from input – to output streams.

We have also shown in Theorem 2 that LSMs can overcome the limitation of a fading memory if one allows feedback from readouts back into the "liquid". Then not only all digital, but (in a well-defined sense) also all analog computers can be simulated by a fixed LSM, provided that one is allowed to vary the readout functions (including those that provide feedback). Hence these readout functions can be viewed as program for the simulated analog computers (note that all "readout functions" are just "static" functions, i.e. maps from $\mathbb{R}^n$ into $\mathbb{R}$, whereas the LSM itself maps input streams onto output streams). In those practically relevant cases that have been considered so far, these readout functions could often be chosen to be linear. A satisfactory characterization of the computational power that can be reached with linear readouts is still missing. But obviously the kernel-property of the underlying "liquid" can boost the richness of the class of analog computers that can be simulated by a fixed LSM with linear readouts.

The theoretical analysis of computational properties of randomly connected circuits and other potential "liquids" is still in its infancy. We refer to [20] for a very useful first step. The qualities that we expect from the "liquid" of a LSM are completely different from those that one expects from standard computing devices. One expects diversity (rather than uniformity) of the responses of individual gates within a liquid (see Theorem 1), as well as diverse local dynamics instead of synchronized local gate operations. Achieving such diversity is apparently easier to attain by biological neural circuits and by new artificial circuits on the molecular or atomic scale, than emulating precisely engineered circuits of the type that we find in our current generation of digital computers, which only function properly if all local units are identical copies of a small number of template units that respond in a stereotypical fashion. In addition, sparse random connections turn out to provide better computational capabilities to a LSM than those connectivity graphs that have primarily been considered in theoretical studies, such as all-to-all connections (Hopfield networks) or a 2-dimensional grid (which is commonly used for cellular automata).

We refer to the 2007 Special Issue on Echo State Networks and Liquid State Machines of the journal Neural Networks for an up-to-date overview of further theoretical results and practical applications of the computational ideas presented in this article.

References

1. W. Maass, T. Natschläger, and H. Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11):2531–2560, 2002.
2. S. Boyd and L. O. Chua. Fading memory and the problem of approximating nonlinear oparators with Volterra series. *IEEE Trans. on Circuits and Systems*, 32:1150–1161, 1985.
3. W. Maass and H. Markram. On the computational power of recurrent circuits of spiking neurons. *Journal of Computer and System Sciences*, 69(4):593–616, 2004.

4. H. Jäger and H. Haas. Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication. *Science*, 304:78–80, 2004.
5. W. Maass, P. Joshi, and E. D. Sontag. Computational aspects of feedback in neural circuits. *PLOS Computational Biology*, 3(1):e165, 1–20, 2007.
6. M. S. Branicky. Universal computation and other capabilities of hybrid and continuous dynamical systems. *Theoretical Computer Science*, 138:67–100, 1995.
7. D. V. Buonomano and M. M. Merzenich. Temporal information transformed into a spatial code by a neural network with realistic properties. *Science*, 267:1028–1030, Feb. 1995.
8. W. Maass and H. Markram. Theory of the computational function of microcircuit dynamics. In S. Grillner and A. M. Graybiel, editors, *The Interface between Neurons and Global Brain Function*, Dahlem Workshop Report 93, pages 371–390. MIT Press, 2006.
9. J. J. Hopfield and C. D. Brody. What is a moment? Transient synchrony as a collective mechanism for spatio-temporal integration. *Proc. Nat. Acad. Sci. USA*, 98(3):1282–1287, 2001.
10. W. Maass, T. Natschläger, and H. Markram. Fading memory and kernel properties of generic cortical microcircuit models. *Journal of Physiology – Paris*, 98(4–6):315–330, 2004.
11. D. Verstraeten, B. Schrauwen, D. Stroobandt, and J. Van Campenhout. Isolated word recognition with the liquid state machine: a case study. *Information Processing Letters*, 95(6):521–528, 2005.
12. C. Fernando and S. Sojakka. Pattern recognition in a bucket: a real liquid brain. In *Proceedings of ECAL*. Springer, 2003.
13. S. Häusler and W. Maass. A statistical analysis of information processing properties of lamina-specific cortical microcircuit models. *Cerebral Cortex*, 17(1):149–162, 2007.
14. D. Nikolić, S. Haeusler, W. Singer, and W. Maass. Temporal dynamics of information content carried by neurons in the primary visual cortex. In *Proc. of NIPS 2006, Advances in Neural Information Processing Systems*, volume 19. MIT Press, 2007.
15. J. Hertzberg, H. Jaeger, and F. Schoenherr. Learning to ground fact symbols in behavior-based robot. In F. van Harmelen ed., editor, *Proc. of the 15th European Conference on Artificial Intelligence.*, pages 708–712, Amsterdam, 2002. IOS Press.
16. K. Bush and C. Anderson. Modeling reward functions for incomplete state representations via echo state networks. In *Proceedings of the International Joint Conference on Neural Networks, Montreal, Quebec*, 2005.
17. M.H. Tong, A.D. Bickett, E.M. Christiansen, and G.W. Cotrell. Learning grammatical structure with echo state networks. *Neural Networks*, 2007. in press.
18. H. Jaeger and D. Eck. Can't get you out of my head: A connectionist model of cyclic rehearsal. In I. Wachsmuth and G. Knoblich, editors, *Modeling Communication with Robots and Virtual Humans*, 2007. in press.
19. P. Joshi and W. Maass. Movement generation with circuits of spiking neurons. *Neural Computation*, 17(8):1715–1738, 2005.
20. O. L. White, D. D. Lee, and H. Sompolinsky. Short-term memory in orthogonal neural networks. *Phys. Rev. Letters*, 92(14):148102, 2004.

This article was processed using the LaTeX macro package with LLNCS style