The “Liquid Computer”: A Novel Strategy for Real-Time Computing on Time Series

Thomas Natschläger
tnatschl@igi.tu-graz.ac.at
www.igi.tugraz.at/tnatschl

Wolfgang Maass
maass@igi.tu-graz.ac.at
www.igi.tugraz.at/maass

Henry Markram
Henry.Markram@weizmann.ac.il
www.weizmann.ac.il/neurobiology/labs/markram/markram.html

Abstract

We will discuss in this survey article a new framework for analysing computations on time series and in particular on spike trains, introduced in [1]. In contrast to common computational models this new framework does not require that information can be stored in some stable states of a computational system. It has recently been shown that such models where all events are transient can be successfully applied to analyse computations in neural systems and (independently) that the basic ideas can also be used to solve engineering tasks such as the design of nonlinear controllers.

Using an illustrative example we will develop the main ideas of the proposed model. This illustrative example is generalized and cast into a rigorous mathematical model: the Liquid State Machine. A mathematical analysis shows that there are in principle no computational limitations of liquid state machines in the domain of time series computing. Finally we discuss several successful applications of the framework in the area of computational neuroscience and in the field of artificial neural networks.

1 Introduction

The analysis of computation in neural systems has often concentrated on the processing of static stimuli. However, numerous ecologically relevant signals have a rich temporal structure, and neural circuits must process these signals in real time. In many signal processing tasks, such as audition, almost all of the information is embedded in the temporal structure. In the visual domain, movement represents one of the fundamental features extracted by the nervous system. Hence it is not surprising that in the last few years there has been increasing interest in the dynamic aspects of neural processing. Processing of real-world time-varying stimuli in real-time is a difficult problem, and represents an unsolved challenge for computational models of neural functions. Simultaneously in computer science several areas such as for example computer vision, robotics, and machine learning have also increased their efforts to deal with dynamic real-world inputs.

Computational models that are commonly used as conceptual basis for the investigation of computations in biological neural systems are Turing machines, finite state machines (automata), and attractor neural networks. These models have in common that they can store bits in a suitable number of stable states, for example attractors (or limit cycles) of an attractor neural network. This capability appears to be less ubiquitous in biological neural systems, since its components exhibit a strong inherent dynamics on several time scales, and in many instances the only stable state is the “dead” state. This observation has motivated us to investigate the question whether there are alternative computational models that do not have to rely on states that remain stable, but rather can be implemented in an environment where all states are transient and are able to process time-varying stimuli in real-time.

In order to approach this question with a fresh mind, one might for example think of carrying out computations in a liquid medium, such as a cup of coffee. As motivated above, the type of computation we want to consider are computations on time series, i.e. computations whose inputs $\boldsymbol{u}(\cdot)$ are functions of time, and whose outputs are again certain functions $\boldsymbol{v}(\cdot)$ of time. Such functions that map input time series $\boldsymbol{u}(\cdot)$ on output time series $\boldsymbol{v}(\cdot)$ are usually called operators in mathematics, but are commonly re-

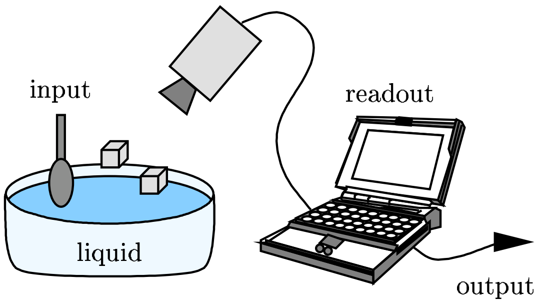


Figure 1: The "liquid computer". The "liquid" (e.g. cup of coffee) gets as input a sequence of perturbations (e.g. spoon hits or sugar cubes dropped). These perturbation are transformed in real-time into "liquid states" (e.g. an image of the surface of the liquid). A memory-less readout (in the sense that it has no access to past states) transforms this liquid states into the desired output time series.

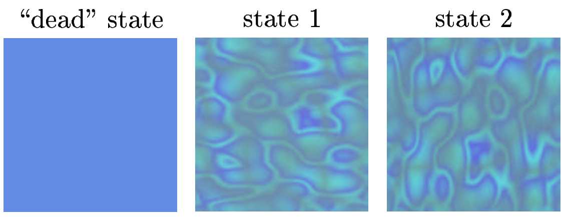


Figure 2: Images of the surface of the liquid are considered as the states of the liquid. State 1 and 2 are generated by two different input time series, i.e. series of perturbations (spoon hits and sugar cubes dropped).

ferred to as *filters* in engineering and neuroscience. We use the term filter in the following.

In the example of computing with a liquid one could for instance view as input time series the sequence of perturbations (for example when a spoon hits the surface of the coffee or sugar cubes that drop into the cup) that continually affect the liquid. The output could be an online (at any time t) classification of the sources of all perturbations that have recently occurred, or a prediction of the next perturbations that are to be expected. The output can for example be computed by a laptop PC from an image of the surface of the liquid which is recorded with a digital camera. We call such a setup the "liquid computer".

Note that the idea of the "liquid computer" is not restricted to the particular choice of the liquid (cup of coffee) and the readout (camera plus PC) given in this illustrative example; this is discussed in more detail in Section 2.

An image of the surface of the liquid taken at time t can be considered as the current state of the liquid or simple *the liquid state* $\boldsymbol{x}(t)$.[1] It is obvious that computations in the "liquid computer" can not relay on stable states since the only stable state of the liquid is the "dead state" (see Fig. 2), and all perturbations of the liquid trigger transient events.

We will argue that for all interesting target filters which one wants to implement with a "liquid computer" it is sufficient to compute the output via a suitable readout mechanism *at any time* t only from the current state $\boldsymbol{x}(t)$ of the liquid. This will enable the "liquid computer" to produce its *output in real-time*: it does not have to wait explicitly until it has gathered enough information to compute the output. In the example of the "liquid computer" shown in Fig. 1 this means: to compute the output at time t the PC only needs to access the image which was taken at time t but no other images taken before time t, i.e. there is no need for the PC to store images, or features of earlier images. In other words: we propose the hypothesis that it is sufficient to apply a *memory-less* device — in the sense that it has no access to states prior to time t — as readout to compute the desired output from the current state $\boldsymbol{x}(t)$ of the liquid.

How can a "liquid computer" implement a desired target filter for which it is definitely necessary to have access to inputs at times $t' < t$ to compute the output at time t (e.g. output at time t the number of sugar cubes dropped into the liquid within the last 2 seconds)? The basic idea is that the current state $\boldsymbol{x}(t)$ at time t of the liquid has to hold all the relevant information about the input. Informally this requires that any two different input time series which should produce different output time series put the liquid into two (significant) different states (see Fig. 2). If the liquid has this property it is at least in principle possible to find a suitable *memory-less* readout (in the sense that it has no access to states prior to time t) to compute the desired outputs in a real-time fashion.[2]

[1]Note that such an image of the surface of the liquid does not describe the full state of the liquid if considered as a dynamical system. The full state of a cup of coffee considered as dynamical system would contain the positions and velocities of all particles within the cup.

[2]Of course a nowadays PC would have enough memory to record a whole stream of images during the last 2 seconds. However, if all relevant information to compute the output $\boldsymbol{v}(t)$ at time t is already contained in

From this point of view one can interpret the readout as a device which retrieves the desired information about the input from the current state of the liquid. Obviously this task of the readout can be accomplished more easily if the states of the liquid contain in a “clearly visible” form the information about the input which the readout is supposed to pick out. Which features of the input are “clearly visible” in the states of the liquid depend on the internal structure of the liquid, for example on the type and range of interaction between the individual elements the liquid is composed of. In particular the internal structure of a liquid determines over which time and spatial scales the inputs are integrated (mixed) and produce the states of the liquid. Hence the internal structure of the liquid determines how useful (for the readout) the states of the liquid are. Unfortunately the internal structure of real liquids like coffee are such that it has rather small time constants for relaxation, and perturbations are propagated only through the very local interaction between molecules. This makes a cup of coffee less suitable as computational device.

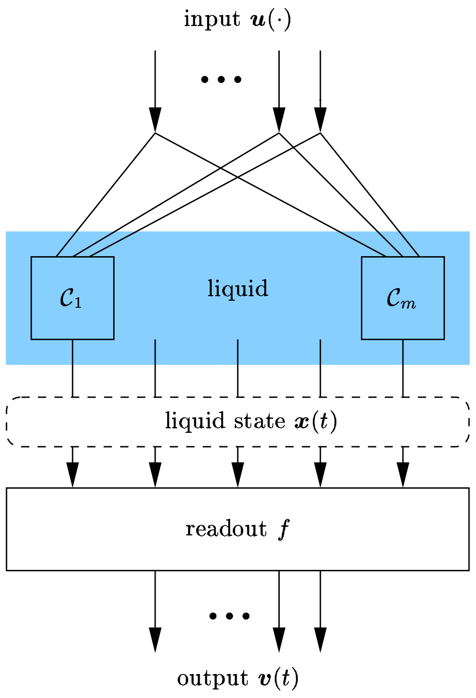
However, neural circuits could constitute ideal “liquids” because of the distributed interactions between its elements (neurons) and the large variety of time scales present in this systems. These characteristic features potentially allow neural systems to function as an optimal integrator of all kinds of sensory input. Optimal in the sense that the state of the neural system serves as a universal source of information about present and past stimuli which can be used by a simple readout mechanism (e.g. a single neuron) to compute any desired output. In fact it was found in [1] by computer simulations that models of small neural circuits can indeed be used as a “liquid” and that readout maps can be trained such that a given task is accomplished. Independently the basic ideas of the “liquid computer” have been investigated in [2] from an engineering point of view. In that work artificial recurrent neural networks (see e.g. [3]) have been used as a “liquid” and simple linear readout functions have been trained to fulfill several different tasks.

the state $\boldsymbol{x}(t)$ it is a) not necessary to store anything about the past and b) wasteful to discard this information since it would probably take more time to compute the output $\boldsymbol{v}(t)$ from several images (about 50 images for 2 seconds of history). Furthermore point b) would imply that the output of the readout can not be given in real-time anymore.

In section 2 we will make these informal ideas about the “liquid computer” more precise, and discuss the general (and more formal) model introduced in [1] for analysing computations on time series: the *liquid state machine* (LSM). We will formulate two simple conditions, the separation property and the approximation property, which - if met - endow any LSM with in principle universal computational power in the time series domain. This analysis shows that devices where the effects of the most common perturbations are transient, can still be computationally powerful. In section 3 we will discuss several successful applications of the LSM approach.

2 The Liquid State Machine

A formal model of a “liquid computer” The *liquid state machine* (LSM) introduced in [1] is a mathematical model of the “liquid computer”. Its basic structure is depicted in Fig. 3.



input $\boldsymbol{u}(\cdot)$
$\mathcal{C}_1$
liquid
$\mathcal{C}_m$
liquid state $\boldsymbol{x}(t)$
readout f
output $\boldsymbol{v}(t)$

Figure 3: A mathematical model of the “liquid computer”: the *liquid state machine.* The liquid transforms the input into liquid states $\boldsymbol{x}(t)$, which are mapped by the memory-less readout function f to the output $\boldsymbol{v}(t) = f(\boldsymbol{x}(t))$.

A LSM consists of some liquid which transforms the input time series $\boldsymbol{u}(\cdot)$ into “liquid states” $\boldsymbol{x}(t)$. A memory-less readout function

f maps the "liquid state" $\boldsymbol{x}(t)$ at time t into the output $\boldsymbol{v}(t) = f(\boldsymbol{x}(t))$. Note that there are no assumptions about the concrete implementation of the liquid and the readout function.

It is simply assumed that the liquid integrates inputs $\boldsymbol{u}(t')$ for $t' < t$ into liquid states $\boldsymbol{x}(t)$ for all times t. From a dynamical systems point of view the liquid is a non-autonomous dynamical system (see [4]) and the time varying function $\boldsymbol{x}(\cdot)$ is the (usually high-dimensional) trajectory of this system. One could think of the trajectory $\boldsymbol{x}(\cdot)$ as the output of an array $\mathcal{C}_1 \ldots, \mathcal{C}_m$ of subsystems of the liquid which define the continuous state vector.

Universal computational power of LSMs We now want to characterize which mappings from input time series to output time series (i.e. which filters) can be computed by LSMs. Can only linear filters be implemented with a LSM or is it also possible to implement complex non-linear filters with an LSM?

In fact it turns out that there are only very weak restrictions on what LSMs can compute: a mathematical theorem [5] states that any *time invariant* filter with *fading memory* can be approximated with arbitrary precision by an LSM.

The condition that only time invariant filters can be computed just exclude such exotic filters where a simple time shift of the input causes not only a time shift of the output but also a different time course of the output. Also the restriction to filters with fading memory is rather weak: it only states that an LSM can not compute a filter that is discontinuous, or whose output strongly depends on infinitely long input histories. Hence one may argue that the class of time invariant filters with fading memory includes practically any relevant filter from a biological as well as from an engineering point of view. From this perspective one could say that LSMs have *universal computational power* for computations on time series.

Only two abstract and conceptually simple properties have to be met to endow LSMs with such universal computational power for computations on time series: The class of components from which the liquid is composed satisfies the point-wise separation property and the class of functions from which the readout maps are drawn, satisfies the approximation property:

Separation Property: All output-relevant differences in the preceding part of two input time series $\boldsymbol{u}_1(\cdot)$ and $\boldsymbol{u}_2(\cdot)$ (before time t) are reflected in the corresponding liquid states $\boldsymbol{x}_1(t)$ and $\boldsymbol{x}_2(t)$ of the system.

Approximation Property: The readout has the capability to approximate any given continuous function f that maps current liquid states $\boldsymbol{x}(t)$ on current outputs $\boldsymbol{v}(t)$.

Implementing specific target filters The preceding results provide a new conceptual framework for analyzing computations on time series, such as spike trains. However they do not yet address the question how to implement an LSM such that it approximates a given target filter. One approach which directly emerges from the LSM model is described in Fig. 4.

1) Choose a suitable liquid
2) Record liquid states $\boldsymbol{x}(t)$ at various time points in response to numerous different (training) inputs $\boldsymbol{u}(\cdot)$
3) Apply a supervised learning algorithm to a set of training examples of the form $\langle \boldsymbol{x}(t), y_u(t) \rangle$ to train a readout function f such that the actual outputs $f(\boldsymbol{x}(t))$ are as close as possible to $y_u(t)$

Figure 4: A general approach for implementing specific target filters with a LSM. $y_u(t)$ denotes the output of the target filter at time t if its input is $\boldsymbol{u}(\cdot)$.

If the liquid fulfills the separation property and the readout function fulfills the approximation property, this procedure leads to an implementation of a LSM which approximates the given target filter. One advantage of this simple procedure is that it is not necessary to take any temporal aspects into account for the supervised learning task, since all temporal processing is done implicitly in the liquid. Another benefit of this procedure is that one can easily implement several target filters in parallel using the same liquid. One just has to train for each target filter a separate readout function.

Despite its simplicity the above procedure has one drawback: it does not specify which liquid and which learning algorithm one should choose to implement a given filter. For example, from a theoretical point of view it would suffice to choose the components of the liquid as a suitable collection of delay lines and to use a quite powerful readout (e.g. a artificial neural network, see e.g. [3] for regarding the details of this approach). On the other hand, one of the empirically most successful approaches in machine learning (support vector machines, see [6]) is based on the observation that almost all practically relevant classification tasks can be carried out by a single perceptron if the inputs are first projected into a very high dimensional space. In the framework of LSMs this implies that if the liquid transforms the inputs into a proper space of liquid states, a very simple readout will suffice to implement the given target filter. Hence there is a trade-off between the complexity of the liquid and the complexity of the readout. The optimal point for this trade-off depends on several factors, such as the type and number of target filter which one wants to implement.

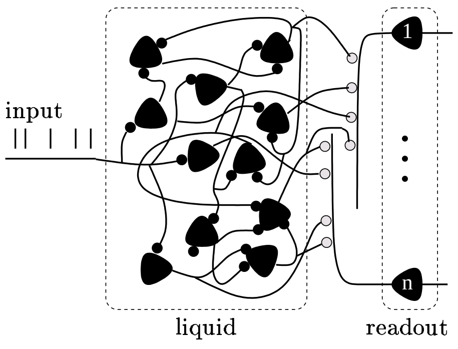


Figure 5: The basic network model investigated in [1]. A randomly connected network of biologically quite realistic model neurons was employed as "liquid". The readout function was implemented by another population of model neurons that received input from all the "liquid-neurons". During training for a particular task only the synaptic strengths of the connections from the "liquid" to the readout (open circles) are modified.

3 Applications of the LSM approach

Computations on spike trains in recurrent neural microcircuits In [1] it was explored how well the LSM approach is applicable to computations in models of neural systems. The main issue investigated was how well small recurrent neural circuits can be understood as a "liquid". A "liquid" in the sense that such circuits function as a universal integrator of a variety of stimuli.

The basic network model used in [1] is shown in Fig. 5. A randomly connected network of biologically quite realistically model neurons was employed as "liquid". The input to this network was via several spike trains. The readout function was implemented by another population of model neurons that received input from all the "liquid-neurons". The fraction of neurons firing around time t was interpreted as the output of the readout. Using the general training procedure described in Fig. 4, a readout function is trained to a specific task by adjusting the synaptic strengths of the connections from the "liquid" to the readout. To accomplish this task a recently developed supervised learning algorithm called p-delta rule was used (see [7] for details about this learning algorithm).

All readout functions trained in this manner used the same recurrent microcircuit (liquid) as (universal) source of information to compute the desired output. The tasks assigned to the different readout maps were chosen such that each one had a focus on another aspect of the input. Results of computer simulations show that such models of neural microcircuits can indeed function as an optimal liquid since all considered tasks can be performed with quit high accuracy.

Furthermore the issue how different connectivity structures of neural microcircuits affect the "usefulness of the liquid" was explored. It turned out that there exists some optimal regime of connectivity parameters for the neural microcircuit. Optimal in the sense that inputs are integrated (mixed) into the liquid state such that the considered computational tasks were performed with maximal accuracy. In other parameter ranges either the inputs are not integrated (mixed) at all and hence no information about past inputs is available in the liquid state, or the circuits exhibit a chaotic behavior. If the circuit exhibits chaotic behavior even the smallest change in the input (which causes only small changes in the target output) causes the liquid to end up in a totally different state, and hence the readout is usually not able to map such totally different states to rather identical outputs.

These approach provides a new platform for investigating through computer simulations the structure and function of neural microcircuits, for example to explore the computational advantages of the extreme complexity and diversity of neurons, synapses, and specific connectivity patterns found in the microcircuity of the brain.

Computations on spike trains in feedforward networks Whereas in [1] recurrent circuits of spiking neurons were considered, in [8] a much simpler type of liquid is investigated; see Fig. 6.

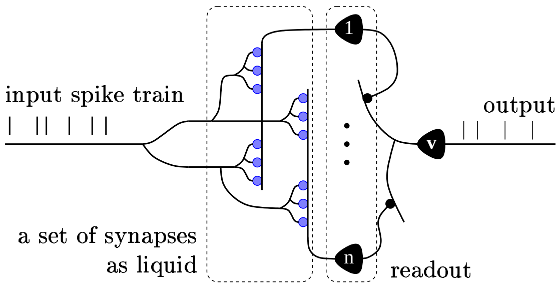


Figure 6: The network of spiking neurons used in [8]. The liquid consists of an array of biologically realistic model synapses. As in [1] the readout is another population of neurons. Its firing activity is explicitly transformed into a spike train by the output neuron v. Networks of this type can be trained to behave like any given small finite state machine.

The liquid was just an array of biologically realistic synapse models. It is known that synapses are quite heterogeneous, i.e. produce quit different outputs for the same input spike train, even within a single neural circuit. In fact these differences, which also vary from synapse to synapse, can serve as short term memory for a neural system, since the amplitude of the postsynaptic response for the current spike contains information about the preceding part of the spike train. Hence the parameters for the model synapses employed as liquid were drawn from distributions in accordance with empirical results (see e.g. [9]). The readout was implemented as another pool of spiking neurons; as in [1]. However, to demonstrate that the LSM approach allows to construct circuits which implement given mappings from input spike trains to output spike trains the firing activity of the readout pool was explicitly transformed into a spike train using an extra single output neuron; see Fig. 6.

The question explored in [8] was to what extent such a simple neural circuit can carry out arbitrarily complex computations on spike trains. It was demonstrated by computer simulations that such a simple neural circuit can be trained to behave like any given small finite state machine. Finite state machines (FSM) are frequently used in computer science as general purpose models for computations on time series. A FSM usually gets a bit-string as input and produces as online output again a bit-string (in our terminology it is a filter which operates on bit strings).[3]

Since the finite state machines which were used as target for the training were drawn randomly, this result demonstrates impressively the generality of the LSM approach. Furthermore these results show that the heterogeneity of synapses enables a simple network of spiking neurons to perform quite complex computational tasks. This might be understood as a hint why the diversity of neurons and synapses found in real neural system may be computationally advantageous.

Computations on analog time series The main idea of the LSM was independently discovered in [2]. The author of [2] considered an artificial recurrent neural network (see e.g. [3]) as liquid.[4]

As a readout function he used a single unit of the same type as the liquid is composed of. Training of this single output unit amounts to a linear regression problem, which is one of the simplest "learning" algorithms one can think of.

The perspective taken is mainly that of mathematics and engineering, where a recurrent network is seen as a computational device for realizing a dynamical system (the liquid). Hence the target filters (learning tasks considered) are from a different nature as presented so far. For example it is demonstrated that the quasi-benchmark task of learning a chaotic attractor can be accomplished by using the training approach depicted in Fig. 4. Another challenging task which could be solved in this way is the design of a controller for a pendulum (private communication; not reported in [2]).

The most appealing features of this learning algorithm for recurrent neural networks (which emerges from the LSM model) compared to others (see e.g. [3]) are its simplicity and robust-

[3]Note that bit-strings can easily converted to spike trains and vice versa. Hence it is possible to compare a FSM and the network depicted in Fig. 6.

[4]The network they used consists of N units. At time step t unit i computes the output $x_i(t+1) = \tanh\left(\sum_j w_{ij}x_j(t)\right)$ where w_{ij} is a parameter describing the connection strength between unit i and j.

ness. These features stem from the fact that only the parameters of a single unit are trained and that this training can be done using linear regression.

4 Summary

We have discussed a novel computational model for analysing computations on time series in particular on spike trains: the "liquid computer" or more formally the liquid state machine. In contrast to models that are inspired by the architecture of the current generation of artificial computing machinery it requires no storage of information in stable states. We have formulated two simple conditions, the separation property and the approximation property, which together endow liquid computers with virtually unlimited computational power in the time series domain.

On a more practical level the LSM approach provides a general method for training a computer model of a neural microcircuit consisting of biologically realistic models for neurons and synapses, to carry out basically any given computation on spike trains. Furthermore this general method can also be applied in the domain of artificial neural networks [2] and yields a simple and robust learning algorithm. Applications of this general method were discussed in Section 3.

To make the work on the LSM approach more accessible to other people we are currently setting up web resources (accessible via `www.lsm.tugraz.at`) regarding data, computer models, and analysis tools for neural microcircuits, which will be functional by fall 2002. In this webpage we will collect data on the anatomy and physiology of neural microcircuits from the lab by Markram (see `http://www.weizmann.ac.il/neurobiology/labs/markram/markram.html`) and others. Furthermore it will provide software for computer models of neural microcircuits that are built according to these data, as well as files with inputs for these circuits and tools for analyzing the circuit outputs. In addition it will offer implementations of learning algorithms by which the readout neurons from these circuits can be trained. In this way the user can not only carry out his/her own computational tests with these circuit models, but can also investigate the merits of the LSM framework.

References

[1] W. Maass, T. Natschläger, and H. Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *submitted for publication*, 2001. electronically available via `http://www.igi.TUGraz.at/tnatschl/psfiles/130.pdf`.

[2] H. Jaeger. The "echo state" approach to analysing and training recurrent neural networks. GMD Report 148, German National Research Center for Information Technology, 2001. electronically available via `ftp://borneo.gmd.de/pub/indy/publications_herbert/EchoStatesTechRep.pdf`.

[3] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation.* Addison-Wesley, 1991.

[4] S. H. Strogatz. *Nonlinear Dynamics and Chaos: With Applications in Physics, Biology, Chemistry, and Engineering (Studies in Nonlinearity).* Addison-Wesley, 1994.

[5] W. Maass and H. Markram. On the computational power of recurrent circuits of spiking neurons. submitted for publication, 2001.

[6] V. N. Vapnik. *Statistical Learning Theory.* John Wiley, New York, 1998.

[7] P. Auer, H. Burgsteiner, and W. Maass. The p-delta learning rule for parallel perceptrons. submitted for publication, 2001.

[8] T. Natschläger and W. Maass. Spiking neurons and the induction of finite state machines. *Theoretical Computer Science: Special Issue on Natural Computing*, 2002. electronically available via `http://www.igi.TUGraz.at/tnatschl/psfiles/fst-learning-tcs.pdf`.

[9] A. Gupta, Y. Wang, and H. Markram. Organizing principles for a diversity of GABAergic interneurons and synapses in the neocortex. *Science*, 287:273–278, 2000.