Chapter 9

# Computer Models and Analysis Tools for Neural Microcircuits

Thomas Natschläger[1], Henry Markram[2] and Wolfgang Maass[1]
[1] *Institute for Theoretical Computer Science, Technische Universität Graz, Austria*
[2] *Brain Mind Institute, Ecole Polytechnique Federale de Lausanne, Switzerland*
*Correspondence to: tnatschl@igi.tu-graz.ac.at*

**Abstract**: This chapter surveys web resources regarding computer models and analysis tools for neural microcircuits. In particular it describes the features of a new website (www.lsm.tugraz.at) that facilitates the creation of computer models for cortical neural microcircuits of various sizes and levels of detail, as well as tools for evaluating the computational power of these models in a Matlab-environment.

**Key words**: neural microcircuits, spiking neurons, computer simulations, real-time computing, learning algorithms, Matlab.

## 1. INTRODUCTION

A major problem in understanding brain-style computing is the identification of the *basic computational unit of the brain* (Zador, 2000). Neurons are obvious candidates. On the other hand there exist numerous emergent properties of recurrent circuits of neurons and synapses which cannot readily be explained on the basis of their components. This has motivated an alternative proposal: that stereotypical neural microcircuits which are repeated throughout the neocortex should be viewed as computational units of the brain (Shepherd, 1988), (Douglas and Martin, 1998). More detailed investigations of cortical neural microcircuits have provided further evidence that these circuits exhibit intriguing stereotypical features, but that these stereotypical circuit patterns are quite complex (Gupta et al., 2000), (Thomson et al., 2002), (Gupta et al., 2002). This additional complexity arises from the fact that stereotypical cortical microcircuits consist of a fairly large number of different types of neurons and synapses, which are combined in quite regular circuit patterns. For computational modeling this provides the additional challenge to explain how computations are organized in such complex microcircuits, and how the diversity of their components and the complexity of their connection patterns might contribute – or even be essential – for the virtually unlimited computational power of biological neural systems. Since at present it appears to be hopeless to analyze the computational power of realistic models for biological neural microcircuits with theoretical methods, computer simulations provide the most promising tool. This chapter discusses publicly available software and analysis tools that appear to be useful for such investigations.

Obviously there exists no agreement regarding the meaning of abstract concepts such as *computation* and *computational power*. In the following we mean with a computation simply an algorithm or circuit that assigns outputs to inputs. Evidence for the computational power of such circuit is provided by the complexity and diversity of associations of outputs to inputs that can be implemented on it. We will focus on the biological most common type of computations where the inputs to a computation are not static batch inputs, like in a computer, but rather *continuous streams of information*, such as for example afferent spike trains from neurons in peripheral neural systems. In addition we focus on the biologically most relevant case of *real-time computing*, where the system cannot afford to wait until a computation has terminated, but instead has to provide an output within a specific time period. More precisely, we focus on the case where the neural circuit has to respond to a stream of inputs with a stream of outputs, where the output at time $t$ represents the best response to the stream of inputs up to time $t$ that the circuit can provide at this time. In terms of concepts from computational theory this shifts the emphasis towards *online computations* and *anytime algorithms* (i.e., algorithms that output at any time the currently best guess at an appropriate response), and away from offline computations that are more common in computer applications (and which have frequently been used as paradigms for computations in biological neural systems).

## 2. WWW.LSM.TUGRAZ.AT

This website, which has been set up by the first author, contains a collection of Matlab[1] programs for constructing and simulating models for neural microcircuits, as well as Matlab programs for generating input streams and for analyzing the resulting circuit output for various benchmark tasks. We have chosen Matlab since it allows easy implementation of the analysis tools that we are using. Furthermore the powerful graphics offered by Matlab makes it easy to visualize the results. One drawback of choosing Matlab is that there is no readily available tool for the actual simulation of neural circuits. We solved this problem by writing our own simulator for leaky-integrate and fire neurons (in C) which can easily be accessed from within Matlab (by means of a MEX interface). However, with a conceptually very simple interface the user can use instead any other simulation software.

### 2.1 The Microcircuit Simulator

We provide a tool for simulating heterogeneous networks composed of leaky-integrate-and-fire neurons, or alternatively of nonspiking (sigmoidal) neurons. This simulator is written in C with an interface to Matlab (there is no standalone version). It is intended to simulate networks containing a few neurons, up to networks with a few thousand neurons and on the order of 100000 synapses.

***Neuron models*** For the case of leaky-integrate-and-fire (LIF) neurons we implemented the standard model where the membrane potential $V$ of a neuron is given by $\tau_m(dV/dt)=-(V-V_{rest})+R(I_{syn}(t)+I_{back})$, $\tau_m$ is the membrane time constant, $V_{rest}$ is the resting membrane potential, $R$ is the input resistance, $I_{syn}(t)$ is the current supplied by the synapses, and $I_{back}$ is a non-specific background current. We have chosen the fixed values $V_{rest}$=0.0 and $R$=1.0. Alternatively the user can offset the voltages and scale the currents by the actual input resistance. If $V$ exceeds the threshold voltage $V_{thresh}$ it is reset to $V_{reset}$ and held there during the absolute refractory period (of length $\tau_{refr}$).
With regard to nonspiking neurons (which may be useful for analyzing population responses in larger circuits) we implemented a model of a sigmoidal neuron with leaky integration. The dynamics of such a neuron is given by $\tau(dx/dt)=-x+g(I_{syn}(t)+I_{back})$ where $x$ is the output (in the range (0,1)) of the unit (which can be interpreted as an average firing rate over time or space), $g$ is the logistic sigmoid function $g(z)=1/(1+\exp(-z))$, and $\tau$ some time

[1] Matlab is a registered trademark of *The Mathworks Inc.*, see www.mathworks.com for more information about Matlab.

constant (see e.g. chapter 7 in (Hertz et al., 1991)).

***Synapse models*** Two types of synapses are implemented: static and dynamic synapses. While for static synapses the amplitude of each postsynaptic current (PSC) is the same, the amplitude of an PSC in the case of a dynamic synapse depends on the spike train that it has seen so far. For synapses transmitting spikes the time course of a PSC is modeled by $A \cdot \exp(-t/\tau_{syn})$, where $\tau_{syn}$ is the synaptic time constant and $A$ is the synaptic strength (weight) which is constant for static synapses and given by the model described in (Markram et al., 1998) for dynamic synapses. For synapses transmitting analog values (such as the output of a sigmoidal neuron) static synapses are simply defined by their strength $A$ (weight), whereas for dynamic synapses we implemented a continuous version of the dynamic synapse model for spiking neurons (Tsodyks et al., 1998).

***Network inputs*** There are two forms of inputs which can be supplied to the simulated neural microcircuit: spike trains and analog currents, see the user manual for details.

***Synaptic plasticity*** The simulator also supports spike time dependent plasticity, STDP, applying a similar model as in (Song et al., 2000). STDP can be modeled most easily by making the assumption that each pre- and postsynaptic spike pair contributes to synaptic modification independently and in a similar manner. Depending on the time difference $\Delta t = t_{pre} - t_{post}$ between pre- and postsynaptic spike the absolute synaptic strength is changed by an amount $L(\Delta t)$. A typical shape for the function $L(\Delta t)$ as found for synapses in neocortex layer 5 (Markram et al., 1997) is shown in Fig. 1. The current implementation allows for an arbitrary choice for the function $L(\Delta t)$, since various different shapes of it have been found for different synaptic connections (Abbott and Nelson, 2000). Synaptic strengthening and weakening are subject to constraints so that the synaptic strength does not go below zero or above a certain maximum value.

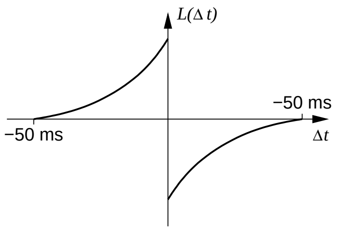


*Figure 1:* Spike time dependent plasticity (STDP) as found for synapses in neocortex layer 5 (Markram et al., 1997).

## 2.2 Tools for Creating Microcircuit Models

When one tries to build a realistic model of a stereotypical cortical microcircuit, one can rely on a number of clear rules that have been established by anatomical and neurophysiological research (Braitenberg et al, 1998), (Gupta et al, 2002), (Thomson et al, 2002). However many of these rules are of a statistical nature, and there are also various aspects where empirical data are still missing. In order to take this into account, the website provides tools not just for building a computer model of one particular neural microcircuit. It also allows the user to specify a probability distribution over neural microcircuits, and to draw random samples according to that distribution. For some questions the evaluation of their average computational power may be more realistic than the analysis of one particular circuit model (that necessarily also has several accidental features). As of this writing there are tools available for constructing multi-column circuits with a distribution of parameters that match those reported in (Gupta et al., 2000).

***Connectivity within a single column*** In a single column the neurons are placed on a $N_x \times N_y \times N_z$ 3-dimensional grid. A certain fraction $f_I$ of these neurons are randomly chosen to be inhibitory neurons. The synapses between these neurons are randomly created in the following way: the probability of a synaptic connection from neuron $j$ to neuron $i$ (as well as that of a synaptic connection from neuron $i$ to neuron $j$) is defined by $C \cdot \exp(-D(i,j)/\lambda^2)$, where $\lambda$ is a parameter which controls both the average number of connections and the average distance between neurons that are connected. $D(i,j)$ is the Euclidean distance between the neurons $i$ and $j$ (which are located on nodes of the $N_x \times N_y \times N_z$ grid). Depending on whether $i$ and $j$ are excitatory ($E$) or inhibitory ($I$), the value of $C$ can be set to different values $C_{EE}$, $C_{EI}$, $C_{IE}$ or $C_{II}$.

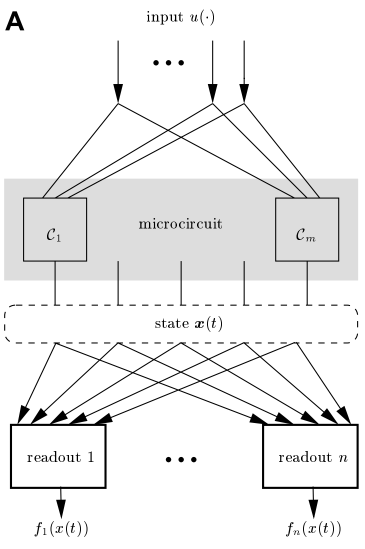
***Synapse parameters*** A synapse can be chosen to be static or dynamic. In either case the associated scaling parameter $A$ (the absolute synaptic strength) also depends on the type ($E$ or $I$) of the pre- and postsynaptic neuron. To be precise: the four parameters $A_{EE}$, $A_{EI}$, $A_{IE}$ or $A_{II}$ describe the mean value of the absolute synaptic strength for a given type of connection ($EE$, $EI$, $IE$, $II$). The actual value of $A_{ij}$ for a synaptic connection from neuron $j$ to neuron $i$ of type $T(ij)$ is chosen randomly from a gamma distribution with mean $A_{T(ij)}$ and a standard deviation of $A_{T(ij)} \cdot SH_A$. Dynamic synapses are described by the additional parameters $U$ (use), $D$ (depression time constant) and $F$ (facilitation time constant) of the synapse model proposed in (Markram et al., 1998), which in general also depend on the type of the synapse. The actual values $U_{ij}$, $D_{ij}$ and $F_{ij}$ for a synaptic connection from neuron $j$ to neuron $i$ of type $T(ij)$ can be drawn from a Gaussian distribution (with appropriately resampled values for outliers) with user-specified mean values $U_{T(ij)}$, $D_{T(ij)}$, $F_{T(ij)}$ and standard deviations $SH_{udf}U_{T(ij)}$, $SH_{udf}D_{T(ij)}$, $SH_{udf}F_{T(ij)}$.

***Intercolumn connectivity*** One can specify separately for each pair of columns the probability of connections between neurons in these two columns and the distribution of the synaptic values for these connections in an analogous manner.

## 2.3 Analyzing the Computational Power of Neural Microcircuit Models

The conceptual framework of a Liquid State Machine (LSM) (Maass et al., 2002) facilitates the analysis of the real-time computing capability of neural microcircuit models. It does not require a task-dependent construction of a neural circuit, and hence can be used to analyze computations on quite arbitrary "found" or constructed neural microcircuit models. It also does not require any a-priori decision regarding the "neural code" by which information is represented within the circuit.

**A**



**B**

1.) Define the neural microcircuit to be analyzed

2.) Record states $x(t)$ of the microcircuit at various time points in response to numerous different (training) inputs $u(\cdot)$

3.) Apply a supervised learning algorithm to a set of training examples of the form $\langle x(t), y_u(t)\rangle$ to train a readout function $f$ such that the actual outputs $f(x(t))$ are as close as possible to the target outputs $y_u(t)$.

*Figure 2.* **A** The Liquid State Machine (LSM). The recurrent microcircuit (liquid) transforms the input into states $\boldsymbol{x}(t)$, which are mapped by the memory-less readout functions $f_1$, ..., $f_n$ to the outputs $f_1(\boldsymbol{x}(t))$, ..., $f_n(\boldsymbol{x}(t))$. **B** A simple procedure for learning a given task defined by its target outputs $y_u(t)$ ($y_u(t)$ denotes the target output at time $t$ if $u(\cdot)$ is given as input).

The basic idea is that a neural (recurrent) microcircuit may serve as an unbiased analog (fading) memory (informally referred to as "liquid") about current and preceding inputs to the circuit.

We refer to the vector of contributions of all the neurons in the microcircuit to the membrane potential at time $t$ of a generic readout neuron as the *liquid state*[2] $\boldsymbol{x}(t)$ (see Fig. 2A). Note that this is all the information about the state of a microcircuit a readout neuron get access. In contrast to the finite state of a finite state machine the liquid state of an LSM need not be engineered for a particular task. It is assumed to vary continuously over time and to be sufficient sensitive and high-dimensional that it contains all information that may be needed for specific tasks.

The liquid state $\boldsymbol{x}(t)$ of a neural microcircuit can be transformed at any time $t$ by a readout map $f$ into some target output $f(\boldsymbol{x}(t))$ (which is in general given with a specific representation or neural code). We will argue that only the synapses of these readout neurons have to be adapted for a particular computational task. This requires that any two different input time series $u(s)$, $s \leq t$ and $v(\mathrm{s})$, $s \leq t$ which should produce different outputs at some subsequent time $t$ put the recurrent circuit into two (significantly) different states $\boldsymbol{x}_u(t)$ and $\boldsymbol{x}_v(t)$ at time $t$. In other words: the current state $\boldsymbol{x}(t)$ of the microcircuit at time $t$ has to hold all information about preceding inputs. If the liquid has this property it possible to train a memory-less readout to produce the desired output at time $t$. If one lets $t$ vary, one can use the same principles to produce as output a desired time series or function of time $t$ with the same readout unit (provided also the states $\boldsymbol{x}(t)$ and $\boldsymbol{x}(t')$ for different times $t$ and $t'$ are different for saliently different input histories).

Fig. 2B outlines the basic procedure for training a readout to perform a given task based on the ideas sketched above. One advantage of this approach is that it is not necessary to take any temporal aspects into account for the learning task, since all temporal processing is done implicitly in the recurrent circuit. Furthermore no a-priori decision is required regarding the neural code by which information about preceding inputs is encoded in the current liquid state of the circuit. Note also that one can easily implement several computations in parallel using the same recurrent circuit. One just has to train for each target output a separate readout neuron, which may all use the same recurrent circuit.

According to the theoretical analysis of this computational model, see (Maass et al., 2002), there are no a-priori limitations for the power of this model for real-time computing with fading memory. Of course one needs a larger circuit to implement computations that require more memory capacity or more noise-robust pattern discrimination. The theoretically predicted

[2] This terminology is motivated by the fact that in contrast to the *finite state* of a finite state machine this circuit output assumes continuous values and varies continuously over time.

universality of this computational model for neural microcircuits can not be tested by evaluating their performance for a single computational task. Instead, each microcircuit model should be tested on a large variety of computational benchmark tasks. Hence it is desirable that many users test the circuit models of the LSM-webpage on their favorite computational problem. Furthermore it seems advantageous to have a set of benchmark tests on which different models can be compared on a qualitative basis.

### 2.3.1 Benchmark Tasks

The tests for analyzing and comparing models of microcircuits that are currently available on the website all have the basic structure outlined in Fig. 2B. Hence to specify a test one just has to define a distribution of inputs and target outputs for these inputs. Then one collects a sufficiently large set of $N_{train}$ training examples by recording the liquid states of the circuit for $N_{train}$ inputs drawn from this distribution together with the associated target outputs. Subsequently a learning algorithm (see section 2.3.2) is applied to this set of training examples, and the performance of the trained readout module is assessed by measuring the error between the actual output and the target function on a distinct set of $N_{test}$ test inputs. In the machine learning community it is common practice that the test inputs are generated by the same distribution as the training inputs.[3] If the error on this test set is small one may conclude that the readout is able to generalize to unseen inputs (and has not simply memorized the training examples, which is called overfitting in the machine learning community. In the following we describe some benchmark tasks which have already been used, and for which code is provided.

***Classification of spike trains:*** The task is to output the class to which a spike train belongs. This task is motivated by the recent work of Hopfield and Brody (Hopfield and Brody, 2001), where they considered the task to classify spatio-temporal pattern of events in time (like spikes). In their case these events in time came from a simple procedure which converts single spoken words into spatio-temporal spike patterns (with at most one spike per channel). Our website provides code for the following generalization of the task: $m$ arrays of $d$ (e.g. $m$=10 and $d$=40 in (Hopfield and Brody, 2001)) Poisson spike trains of frequency $f$ and length $T_{max}$ (e.g. $f$=20Hz, $T_{max}$=0.5sec) are generated, and fixed as (spatio-temporal) templates 1 to $m$. For $i$=1 to $m$ one generates jittered versions of template $i$ by varying each spike in template $i$ by a random drawn amount given by a Gaussian distribution with zero mean and a given STD; this STD is called *jitter*. The

[3] However the software lets the user specify also a distinct distribution.

task is to output the number of the template from which the spike train was actually generated.

***Classification of segments of jittered spike trains:*** This is a more complex task which can be used to evaluate the fading memory capability of a specific microcircuit model. The goal is here to output the class to which a specific *time segment* of the input spike trains belongs. The input distribution is characterized as follows: All inputs are of length $T_{max}$ (e.g. 1.0sec) and consist of $n$ (e.g. 4) segments of length $T_{max}/n$ each. For each segment $m$ (e.g. $m$=2) templates are generated randomly ($d$ Poisson spike trains with a frequency $f$ as described in the preceding paragraph). The actual input spike trains are generated by choosing for each segment one of the $m$ associated templates, and then generating a jittered version of it. One has in general $n$ readout modules, whose task is to output for each of the $n$ segments the number of the template from which the corresponding segment of the input was generated.

Fig. 3 shows the results for a microcircuit with $\lambda$=2 when tested on this task ($d$=1, $n$=4, $m$=2, $f$=20Hz, $T_{max}$=1sec). Two cases were investigated: a) with static synapses in the microcircuit and b) with dynamic synapses in the microcircuit. Note that for the case of static synapses the liquid state of the microcircuit contains substantially less information about preceding input segments than with dynamic synapses.

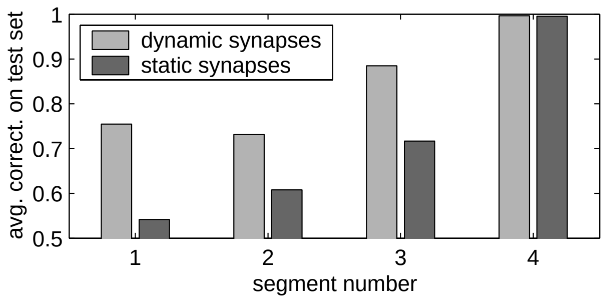


*Figure 3:* A result for the task "Classification of segments of jittered spike trains". 4 readout modules $f_1$ to $f_4$, each consisting of a single perceptron, were trained for this task. The readout module $f_i$ was trained to output 1 at time $t$=1sec if the $i^{th}$ segment of the previously presented input spike train had been constructed from the corresponding template 1, and to output 0 at time $t$=1sec otherwise.

The avg. correctness (percentage of correct classification on an independent set of 500 inputs not used for training) is calculated as the average over 50 trials. In each trial a randomly connected circuit was constructed (1 column, $\lambda$=2) and the readout modules $f_1$ to $f_4$ were trained for perturbed versions ($jitter$=4ms) of randomly chosen Poisson spike trains

($f$=20Hz) as templates. Such averaging is necessary to get statistically significant results. On the other hand a lot more CPU time is needed to get the simulations done. Since such trials are all independent they can easily be done in parallel. The benchmark tests provided at www.lsm.tugraz.at support this type of parallelism; see subsection 2.3.3.

***Retrieval of delayed sum of rates:*** This is a more challenging memory task: at time $t$ output the sum of rates of the input spike trains, averaged over a past time window. The input to the liquid consists of $d$ Poisson spike trains. The rate of theses spike trains is modulated via a randomly chosen function. More precisely each input spike train has an instantaneous rate of $f_{max}\cdot r(t)$ where the random drawn function $r(t)$ is restricted to the interval (0,1). The function $r(t)$ is a sum of several frequency components $f_i$ with independently chosen amplitudes and phase shifts; actually we choose the amplitude $a_i$ of the sin part and the amplitude $b_i$ of the cosine part for the modulation frequency $f_i$ from a Gaussian distribution with zero mean and a variance of $\sigma$=1. In summary, $r(t)$ is of the form $r(t) = Au(t) + B$ with $u(t) = \Sigma_i\, a_i \sin(2\,\pi\, f_i\, t) + b_i \cos(2\,\pi\, f_i\, t)$ where $A$ and $B$ are some constants chosen such that $r(t)$ is in the range (0,1). The task is then to output at time $t$ the sum of rates (normalized to the interval (0,1)) averaged over the interval ($t$-$D$-$W$,$t$-$D$) where $W$ is the length of the interval and the "delay" $D$ specifies how far in the past this interval lies. Fig. 4 shows results for two different microcircuits and for different values of the "delay" $D$. Note the significantly worse performance of the microcircuit with very sparse connectivity ($\lambda$=0.2). This shows that recurrent connections are necessary for the microcircuit to (temporally) integrate the stimulus.

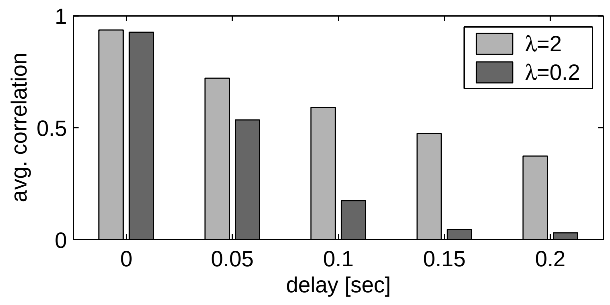


*Figure 4:* Results for the "Delayed Sum of Rates" task. 5 readout modules $f_0$, $f_{50}$, $f_{100}$, $f_{150}$, and $f_{200}$ were trained (via linear regression) to map the liquid state of the circuit onto the scaled sum of rates for the five corresponding different values of $D$: 0ms, 50ms, 100ms, 150ms, and 200ms. The performance is measured as the average correlation between the actual output and the target output on an independent test set.

### 2.3.2 Available Learning Algorithms

All the previously described benchmark tasks employ some supervised learning algorithm in order to train the readout to map the liquid states of the microcircuit to the desired output. Note that in a biological context the target outputs may be provided by the environment (for example in the case of prediction tasks, or for function approximation in reinforcement learning), so that the learning algorithm is supervised only from the point of view of the local circuit, but unsupervised from the point of view of the system. In principle any supervised learning algorithm can be used for training readout modules. In fact it is rather straightforward to add a new algorithm to the set of readily available ones. In this section we will shortly describe the learning algorithms whose code is currently available on the website. The focus lies on learning algorithms that are commonly discussed in the context of neural computation. We do not go into the details of a particular algorithm, but just want to highlight some pros and cons. Excellent sources regarding details of these learning algorithms (except for the p-delta rule) are (Duda et al., 2001) and (Hertz et al., 1991).

***The delta rule*** is one of the oldest and most studied neural learning algorithms. It is designed to train a single perceptron (or threshold gate) for a certain task. Due to the binary output of a perceptron it is limited to classification tasks. The appealing features of this algorithm are its simplicity and its online character which means that learning can be done on an example by example basis and it is not neccessary to have the whole batch of training examples available at once. The disadvantage of the delta-rule is that it does not converge to a stable solution if there is no perfect solution of the task at hand. (i.e. if the data is not linearly seperable).

***Support vector machines (SVM)*** A SVM[4] is a more advanced algorithm for finding the parameters (weights) of a single perceptron applied to a virtual nonlinear projection of the data into a very high dimensional space (see (Vapnik, 1998) for details. The main idea is to choose among all the possible weight settings one which is least likely to overfit. An advantage of a SVM is that it is guaranteed to find the optimal weight setting (by means of a quadratic programming approach). The price to pay for this is the large computation time if the dimension and the size of the training data are substantial .

***The p-delta learning rule*** The p-delta rule is a generalization of the delta

[4] In the case of a nonlinear SVM the input is first nonlinearly transformed in a high-dimensional feature space where then the optimal weight vector is determined.

rule that trains a population of perceptrons to adopt a given population response (in terms of the number of perceptrons that are above threshold). In contrast to other distributed neural learning algorithm it requires very little communication between the neural units (for details see (Auer et al., 2002)). The resulting readout function can also be implemented as a pool of unconnected spiking neurons. Obviously *n* perceptrons can output not only a binary value but a number between 0 and 1 (in discrete steps of size 1/*n*). In fact, pools of perceptrons have the universal approximation property, i.e., they can approximate any given continuous function on any compact domain with any desired degree of precision. The p-delta learning algorithm is in some respect similar to support vector machines, since it explicitly tries to avoid overfitting. One possible drawback is that the user has to get a feeling for the numerous parameters which one has to set. However, by using the default parameters one usually gets quite good results.

***Linear regression (least mean square)*** Linear regression is probably one of the simplest "learning" algorithms. It is the algorithm which one should try first since it is fast, there are no parameters to tune, one knows exactly what is computed and if one has enough data overfitting is unlikely. Furthermore it can be applied to binary (the output is simply passed through a threshold operation) as well to analog target functions

***The back-propagation algorithm*** The back-propagation algorithm is a very popular algorithm for training multi-layer feed-forward artificial neural networks. Like the p-delta rule it has the advantage that it can in principle approximate very complicated target functions if one chooses the right network architecture (which is not a trivial problem). However, one has to be careful not to use too large networks in order to avoid overfitting. Regarding more details about back-propagation we refer the reader to (Hertz et al., 1991) and (Duda et al., 2001).

***Which learning algorithm should I use?*** The answer strongly depends on the type of microcircuit that one wants to evaluate, the task one wants to implement, and last but not least on the type of results one is looking for. If one is able to get good performance with a very simple readout such as a simple perceptron (e.g. trained with the delta rule or linear regression) one can infer that the microcircuit not only carries out all temporal integration that is needed, but also projects its information about preceding inputs into a sufficiently high dimensional space to facilitate subsequent linear pattern recognition (in other words: in this case the microcircuit implements in addition a sufficiently nonlinear and high dimensional kernel, which plays a similar role as the kernel of a SVM).

### 2.3.3 Support for Parallel Processing

The software supports parallel execution of independent parts of the benchmark tasks. Such independent parts include simulation of a given microcircuit with different stimuli and training of several readout function for the same set of inputs (like in Fig. 3 and 4). Note that a single microcircuit will always be simulated at one processor. The following setup is required to be able to utilize the parallel processing feature: a cluster of Unix machines (we use Linux, Redhat 7.2 and SuSE 8.0) which share a common file system (via NFS) and where the software packages Matlab, the Parallel Virtual Machine (PVM) Library (http://www.csm.ornl.gov/pvm/), and the Parallel Matlab Toolbox (Svahn, 2001)[5] are installed.

## 2.4 Contributing to www.lsm.tugraz.at

The microcircuit models, benchmark tasks and learning algorithms described so far should just serve as a starting point for this website. We would like to encourage everybody to contribute his/her models, simulation tools, data, algorithms and last but not least links to related webpages to www.lsm.tugraz.at.

As of this writing there is no automated way for doing this. But any material which is potentially related/interesting can be sent by e-mail (lsm@igi.tu-graz.ac) to the maintainers of the website.

## 2.5 Comparison to Other Tools for Analyzing Neural Microcircuits

There are many other tools available for the simulation and analysis of the type of networks which are currently supported via the software available at www.lsm.tugraz.at. A quite comprehensive list can be found at www.hirnforschung.net/cneuro/cneuro_software.htm. The goal of most of these tools is to provide some kind of general purpose simulation and analysis environment. In principle it would have been possible to implement all code provided at www.lsm.tugraz.at for example within GENESIS (www.genesis-sim.org) or NEURON (www.neuron.yale.edu), to name two prominent representatives of this class of simulators. The advantages would have been that these systems are designed to simulate biologically realistic neurons and one does not have to worry about the particularities like numerical integration. However, we found that most of these systems have disadvantages in comparison with the standard tool Matlab when one

[5] Note that each Matlab process which is spawned uses one full Matlab licence.

implements the analysis tools (for example the learning algorithms) that are needed to evaluate the computational power of microcircuit models. Another (not major) issue was simulation speed: general purpose simulators are usually slower than specialized tools (due to the overhead needed for generality).

The software describe in this chapter is limited to moderate sized models and does not scale to really large models. One project aiming for the simulation of large scale models is the NeoCortexSimulator developed at the Goodman Brain Computation Lab (see http://brain.cs.unr.edu). The goal of this project is to emulate a multi-columnar brain of up to 1 million compartmental neurons by the year 2003. To achieve this performance the simulation software makes use of the supercomputing Beowulf network of the University of Nevada, Reno.

## 3. OUTLOOK

As soon as more experimental data regarding the connectivity and anatomy of neural microcircuits will become publicly available, we plan to add software to the website for creating neural microcircuits that reflect these new data. However for building such more complex microcircuit models it may turn out that one needs a more abstract language for describing the neural microcircuit. One project which tries to establish such a description language for neural systems (which is independent of the software which is actually used to simulate the circuit) is NeuroML, see http://www.neuroml.org.

In the near future we will provide on our website additional analysis tools that are based on concepts from information theory. These will complement the application of learning algorithms by a direct analysis of the mutual information between the liquid state (or other internal variables) of a microcircuit with the preceding circuit input. The estimation of this mutual information is not a trivial problem, since one has to deal with the problem of undersampling (see chapter 10: A practical guide to information analysis of spike train by S. Panzeri, R.S. Petersen and S.R. Schultz). However we think that this is a valuable way to analyze functional properties of neural microcircuits.

We also plan to make publicly available new software which allows the application of microcircuit models to simple sensory processing tasks, such as the ones discussed in (Legenstein et al., 2002). There neural microcircuit models were trained to predict movements of different objects, that move within an unlimited number of combinations of speed, angle, and offset over a simulated visual field. This approach provides interfaces for using simulated models of biological neural microcircuits in actually sensory processing tasks, for example for controlling a mobile robot.

The software currently available is based on the extremely simple LSM architecture shown in Fig. 2A. For a biologically more realistic scenario we will also allow for feedback from the readout back into the microcircuit. In fact, neurons within the liquid circuit may simultaneously serve as readout neurons, thereby blurring the distinction between a microcircuit and its readout. To explore this more complex scenario we are currently developing the necessary software tools. Another simplification made so far is that we are only looking at one microcircuit – more precisely one microcircuit and its associated readouts - at a time. In the spirit of the discussion about microcircuits as potential computational units of the brains it is obvious that one also wants to look at hierarchies or even recurrent networks of such units. Software for such higher level networks also needs to be developed.

Last but not least some kind of graphical user interface (GUI) is needed, which should make the available tools more easily accessible. We are looking forward to feedback and software contributions from the users.

## REFERENCES

Abbott, L. F., and Nelson, S. B. (2000) Synaptic plasticity: taming the beast. *Nature Neurosci.* 3, 1178-1183.

Auer, P., Burgsteiner, H., and Maass, W. (2002) Reducing communication for distributed learning in neural networks. *Proc. ICANN'2002*. Online available as # 127 on http://www.igi.tugraz.at/maass/publications.html.

Braitenberg, V., and Schuez, A. (1998) *Cortex: Statistics and Geometry of Neuronal Connectivity*, 2nd ed., Springer Verlag, Berlin.

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

Duda, R. O., Hart, P. E., and Stork, D. G. (2001) *Pattern Classification*, 2nd ed., John Wiley & Sons, New York.

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

Gupta, A., Silberber, G., Toledo-Rodriguez, M., Wu, C. Z., Wang, Y., and Markram, H. (2002, in press) Organizing principles of neocortical microcircuits. *Cellular and Molecular Life Sciences*.

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

Hopfield, J. J., and 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.

Legenstein, R. A., Maass, W., and Markram, H. (2002) Input prediction and autonomous movement analysis in recurrent circuits of spiking neurons, submitted for publication. Online available as # 140 on http://www.igi.tugraz.at/maass/publications.html.

Maass, W., Natschläger, T., and Markram, H. (2002, in press) Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*. Online available as # 130 on http://www.igi.tugraz.at/maass/publications.html.

Markram, H., Lubke, J., Frotscher, M., and Sakmann, B. (1997) Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. *Science* 275, 213-215.

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

Markram, H., Ofer, M., Natschläger, T., Maass, W. (2002, in press) Temporal integration in neocortical microcircuits. *Cerebral Cortex*.

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

Song, S., Miller, K., and Abbott, L. F. (2000) Competitive Hebbian learning through spike-timing-dependent synaptic plasticity. *Nature Neurosci*. 3, 919-926.

Svahn, E. (2001) *Parallel Matlab Toolbox: User Documentation.* Masters Thesis, Chalmers University of Technology, Sweden. To get a copy of the toolbox contact E. Svahn (email: ersva@igb.polymtl.ca, d96svahn@dtek.chalmers.se) or get it via ftp://ftp-at.e-technik.uni-rostock.de/pub/pm/.

Thomson, A., West, D. C., Wang, Y., and Bannister, A. P. (2002, in press) Synaptic connections and small circuits involving excitatory and inhibitory neurons in layers 2 to 5 of adult rat and cat neocortex: triple intracellular recordings and biocytin-labelling in vitro. *Cerebral Cortex*.

Tsodyks, M., Pawelzik, K., and Markram, H. (1998) Neural networks with dynamic synapses. *Neural Computation* 10, 821-835.

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

Zador, A. (2000) The basic unit of computation. *Nature Neurosci.* 3, 1167.