On real-world temporal pattern recognition using Liquid State Machines

Jilles Vreeken

Intelligent Systems Group,
Institute for Information and Computing Sciences, Utrecht University
Correspondence e-mail address: jvreeken@cs.uu.nl

Universiteit Utrecht

University of Zurich

Submitted in partial fulfilment of the requirements
for the degree of Master of Science

Student
Jilles Vreeken

Period
September 2003 – June 2004

Supervisors
Intelligent Systems Group
Utrecht University, the Netherlands
Prof. Dr. John-Jules Ch. Meyer
Dr. Marco A. Wiering

Artificial Intelligence Laboratory
University of Zurich, Switzerland
Simon Bovet, M. Sc.

Abstract

We investigated the applicability of the recently introduced Liquid State Machine model for the recognition of real-world temporal patterns on noisy continuous input streams. After first exploring more traditional techniques for temporal pattern classification, we provide a brief introduction of spiking neuron models. These can be used as the dynamic 'liquid' filter in the LSM model that pre-processes continuous input such that relatively simple linear readout units can extract information of recent events. With a dedicated set of experiments we show that these networks do integrate important temporal features and that LSMs can be applied for fixed-moment recognition of very noisy temporal patterns with high performance. Continuous extraction of information is possible, but very dependent on the presentation of the liquid state to the readout units. We created a method that enabled us to transform rough recordings of artificial whisker sensors to spike train probability templates, which we used successfully for tasks of real-time texture recognition. The application of multiplicative spike-timing dependent plasticity to a network with dynamic synapses led to significantly higher classification performance than we could attain with static networks.

Acknowledgements

This research project is a direct result of the many opportunities provided by Adaptive Intelligence Laboratory at Utrecht University and the Artificial Intelligence Laboratory at the University of Zurich. I would like to thank the many people that contributed either directly or indirectly to this project sincerely and wholeheartly for helping in making it a success.

First of all, I'd like to express my gratitude to prof. dr. Rolf Pfeifer for making the AI Lab Zurich a great place to do research. My four-month stay there would never have been the same without my supervisors Simon Bovet and Miriam Fend and the valuable discussions and ideas we shared.

Most importantly, though, I would like to thank prof. dr. John-Jules Ch. Meyer and dr. Marco Wiering for starting and maintaining the Utrecht University Robolab, their interest in robotics and artificial intelligence and their open attitude for research beyond their own specializations. They allowed me to co-found the Adaptive Intelligence Laboratory, which lead to a many successful research projects and fruitful collaborations.

This would've never been possible without fellow students Matthijs van Leeuwen and Arne Koopman. Apart from starting the lab, sharing a great form of humour and bouncing many ideas, we had lots of fun pushing each other to higher levels of performance. Thanks, guys.

Last, but certainly not least I would like to thank my family for constant support and keeping up with someone saying artificial and neuron many times more than normally would be considered sane.

Table of Contents

Chapter 1
Introduction

Real-world temporal pattern recognition is one of the most popular activities in the world. In fact, unless you're pushing up daisies you're doing it right now and all day long. Whether it's recognizing the song of a bird, catching a ball or reading a thesis, we're always combining the information we gather over time to extract important features and find possible meaning. If you think about it, for a great many types of tasks we simply need to have information over time, single snapshots of a 'now' often don't suffice. Apart from visual recognition of faces and objects, many patterns in the real world around us only present themselves over periods of time, showing their features and characteristics over different 'now's. Our senses provide us with endless streams of information about the world and our brain is very good at detecting interesting features and patterns over completely different time scales.

Before we can happily start dealing with something as 'real-world pattern recognition' we first have to define what a pattern is at all. Intuitively we define it as a more-or-less reliable set of characteristics of a thing, group, abstract feature or concept. This makes that the act of pattern recognition consists of identifying such a person or thing by some or all of the characteristics in the information we have about it. The tricky part here is that these 'characteristics' can be virtually anything in the form, shape and values of that data we're dealing with. Also, there are many types of data, ranging from nicely constrained statistical information gathered via questionnaires or supermarket customer cards to raw sensory input acquired directly from the real world. As often is the case, less is more. Generally speaking, having a lot of characteristics in the data makes matters worse; it makes it harder to find just those stable features that help us in identifying patterns reliably.

Though static pattern recognition is already quite difficult from a computational point of view, detecting patterns over time is a complete different story. Most good pattern recognition techniques available today are confined to snapshot recognition: they literally attempt to recognize a pattern in one time-slice of input space, basically a Polaroid picture of the current input. This of course works great for problems where all necessary information is in the still image, like for instance face-recognition (see fig. 1). However, there are obviously inherent problems with this approach when we want to try to recognize a pattern in an input sequence.

Figure 1. Face recognition is generally a form of static pattern recognition; extracting features from single still images and seeing whether these match a template. While this is already difficult, extracting information from series of stills is much more complex.

This is exactly where the catch is, as many of the patterns we would like to recognize are temporal of nature. Audio, for instance, obviously is: the current frequency or voltage from a microphone doesn't tell us much about the sound, let alone which word was possibly uttered. The same goes for visual data, unlike recognizing a face, movement can't be detected based on a single still. In order to determine these kinds of characteristics it is necessary to relate the current input to former ones: to have some form of short-term memory.

There are quite a number of methods that work well for detecting repeating features in single snapshots, but there are very few methods that do the same equally well for patterns over time. Even fewer deal well with noisy data. As the world around us is particularly noisy, for real-world pattern recognition we need methods that are able to cope with the inherent noisy data that comes from it. Biological systems deal with noise very well and so do systems that mimic nature like artificial neural networks. However, there are serious problems with these computational techniques when it comes to training and controlling their dynamics when dealing with input over time.

1.1 Liquid State Machines

Recently the model of the Liquid State Machine (LSM) has been presented [48]. It is a novel approach that theoretically allows for real-time computation on continuous input streams in parallel. Randomly generated networks of neurons with realistic biological properties can be used for pre-processing inputs (see fig. 2). Radically different from other approaches is that these networks do not have to be trained for specific tasks. Their network dynamics create unique state trajectories that provide a fading short-term memory of the recent inputs. Relatively simple techniques can be used to extract features from the 'liquid' state of these networks. The readout units can be trained not just to extract features, but can also be used as a mechanism for temporal pattern recognition [48]. The system is partly biological plausible, as parts of the cerebral cortex have been identified to perform sensory integration tasks in small and homogenous 'columns' of neurons [38]. It is unknown, however, how the brain teaches itself to extract features from these dynamic networks. All together, the framework provides a new and promising approach for feature extraction and temporal pattern recognition on continuous input streams.

Figure 2. A Liquid State Machine uses an excitable medium to pre-process low-dimensional inputs, so that simple readout units can extract temporal features. Typically complex recurrent neural networks are used, but its function resembles a tank of liquid: as the inputs disturb the surface they create unique ripples that propagate, interact and eventually fade away. After learning how to read the water's surface we can extract a lot of information about recent events, without having to do the complex input integration ourselves. Real water has successfully been used for this in a task of speech recognition [22].
Image © Pennon Group UK.

Though there recently have been a flurry of publications on Liquid State Machines [5,22,26,28,36,37,48,50,51,58], much is still unknown about what can be expected of their computational power. Theoretically all sorts of complex systems can be applied as pre-processors, but little is said about what systems would perform this task best. Of the systems that have been used it is not even clear what parameters and generation methods allow for the best information pre-processing. Also, nearly all results published have either been focusing on details of the framework, or were aimed at applying them for seemingly complex tasks. Though strong claims have been made about the computational strengths, it must be said the real complexity of these tasks is hard to estimate. Effectively, still little is known about the applicability and real computational powers of liquid state machines with spiking neural networks as temporal input pre-processors.

1.2 Thesis outline

The main goal of this thesis is to identify how well Liquid State Machines can be used for recognizing temporal patterns in noisy continuous input streams. We will start with an overview of existing techniques where we will quickly cover methods as chunking and tapped delay lines, hidden Markov models and recurrent neural networks. All these techniques have their individual strengths and weaknesses for application on this particularly complex task. We will pay extra attention to the feature that allows for fair comparison with the Liquid State Machine framework.

Biological neural networks are proven to excel at noisy temporal pattern recognition [38]. We offer a short introduction to spiking neuron models, a class of artificial neurons that is much more biologically realistic than traditional neural networks [63]. Theoretically spiking neurons are also computationally a lot more powerful as they are able to react non-linearly to individually timed inputs [45]. Apart from general theory on the strengths of pulse coding opposed to more traditional rate coding, we will cover the well-studied and conceptually simple model of integrate-and-fire neurons [25]. Using dynamic synapses that allow for activity bursts and input habituation can further increase the level of network realism [47]. However, exactly all these non-linear reactions to input make the dynamics of spiking neural networks very hard to control [7]. As no generally applicable methods for the supervised training of these networks have been found yet, we will zoom in on the interesting effects of applying unsupervised spike-timing dependent Hebbian learning for creating network homeostasis [1].

Chapter 4 describes the Liquid State Machine model formally and shows that it has a near-identical twin: the Echo State Machine [34]. Both use the dynamics of complex recurrent neural networks for pre-processing temporal input and train additional mechanisms for extracting information from snapshots of the dynamic state of these networks. Though both have been tested in a variety of different tasks, little is known about their applicability on tasks in general and temporal pattern recognition in particular.

After a detailed description of our methods of experimentation, we provide the results of our series of experiments that investigate the applicability of the Liquid State Machine for temporal pattern recognition on single input streams. These experiments have been set up to provide insight in the capabilities power of the framework for the recognition of complex patterns in realistically noisy continuous input streams. Apart from doing this for controllable template generated spike trains we'll also test whether we can apply this method for realtime texture recognition. In the last chapters we offer a detailed analysis of our findings, relating it to other work and pinpointing important aspects for future investigation.

Chapter 2
Temporal pattern recognition

The world around us is extremely dynamic, everything changes continuously over time. Whether it's growth, development or just plain physics, we're surrounded with change. Nature equipped us very well to deal with it. We're brilliant at detecting and coping with these changes, even as they do occur over completely different time scales. Trees and crop grow, people grow older, leaves wave in the wind and sounds are propagating waves in the air. Even though all of these examples are of completely different categories, they are all prime examples of events that can only happen over time.

We are so accustomed to time and it's accompanying change, that it's easy to forget how complex it actually is to deal with. Also, it's easy to loose out of sight that it's not just us that are so good at it, but all life forms within their niche. Researchers in robotics [3] and artificial intelligence [55] know this perfectly well, however. Building something that can operate autonomously in an intelligent manner in the noisy and ever-changing environment of the real world is a task we've not yet succeeded in. Uncertainty caused by noise is a major pain in the behind. But dealing with sensory input over time is most certainly its equal in causing us trouble.

Time is what sets temporal pattern recognition aside from the normal blend we introduced in the previous chapter. Albert Einstein once wrote 'The only reason for time is to prevent all things from happening at once'. Apart from being a famous quote it pinpoints the very reason why temporal pattern recognition is so damn difficult [60]. Instead of having all characteristics to base your decision on at once, they are spread out over many slices of time. A single freeze-frame of time is often of little or no information; in some way a series of these stills has to be collected to detect the pattern we're looking for. In other words, we need a form of memory that spans enough time to hold the key characteristics of our pattern if we want to endeavour in temporal pattern recognition.

Speech recognition is a good example of finding patterns in time: high-level features as phonemes and words are to be extracted from the low-level audio recordings. While still being a very daunting task, it does have a certain number of properties that make it simpler than many other dynamic real world applications. Though everybody's pronunciation of words is different, there are identifying properties likely to be present [31,32]. These properties range from the moments on which frequencies rise, peak and fall to the relative strength at which they are being created. We can use filtering techniques to extract these features, or allow the recognition method to discover these relations for itself. Many tasks do not have such apparent and easily extractable features for identification, one of which is the recognition of textures using whiskers [27].

2.1 Adding memory

Both many statistical methods and standard feed-forward neural networks do not possess real memory. These methods provide only a direct mapping from input to output. In their basic form they are therefore unsuited for temporal pattern recognition. However, we can patch these techniques up by simply combining various stills into a bigger picture: an input window that hopefully contains enough information for detecting the desired pattern. Statistical methods often apply chunking to the input stream, analysing each block separately. For speech-recognition this often includes pre-processing the chunks of recorded speech with techniques like power-spectrum analysis and Fourier transformation. The analysis of these sound-chunks helps in determining whether the atoms of spoken words, phonemes, are present. Feed-forward neural networks form another class of very potent classification techniques that can be presented with a window of inputs. A simple method for enabling these to process time-series is the usage of tapped-delay lines. Their whole function can be explained simply as instead of just the last input, these networks get presented all inputs from the last *n* time-slices [18].

These methods effectively force a form of short-term memory into the input-space of the basic recognition method. It should, however, come as no surprise that this has a cost. The first problem is that the input-space can grow enormously if patterns that span a relatively long period are to be detected [60]. Another major problem is to determine the required size of this input-window: it should hold enough information for the pattern to be recognized, but not be too large either. The optimal window size is often not known beforehand [33].

Processing a lot more input increases the required computational power, but larger input-spaces pose more of a problem for the ability of solving the task at all [60]. Though we can relatively easily scale up computing power, there’s no way we can control the exponential increase of possible correlations between inputs and (desired) outputs. Finding just those configurations of characteristics that indicate the sought pattern could work fine if we didn’t have to worry about noisy data, but unluckily for us we don’t live in such a world. These problems exist for all methods for temporal pattern recognition, but have a particular grave effect on these patch-ups, as these require a direct mapping from input [18].

2.2 Hidden Markov models

It would be rather beneficial if we would not have to present a full series of inputs for analysis, but would let the method have its own form of memory. This allows for internal pre-processing of input states, making it possible that longer-term casual effects and levels of noise are picked up and dealt with within the decision process. A good example of a state-based decision method is a hidden Markov model [15,57]. It has been the standard method for speech recognition since the 1970s.

A hidden Markov model works by constructing a mathematical structure called a (hidden) Markov chain. This is a full state model of the input space it is likely to encounter, see fig. 3. Each transition between states is associated with a random process or simply a probability for each type of input class. These input classes are what we want to extract from the data, we would like to find the hidden (hence the name) generator model. By feeding in the input the model tells us which transitions would be most probable for what generator. By combining each of these proba-bilities for each generator we get an approximation of the chance it was used in generating the full stream [16].

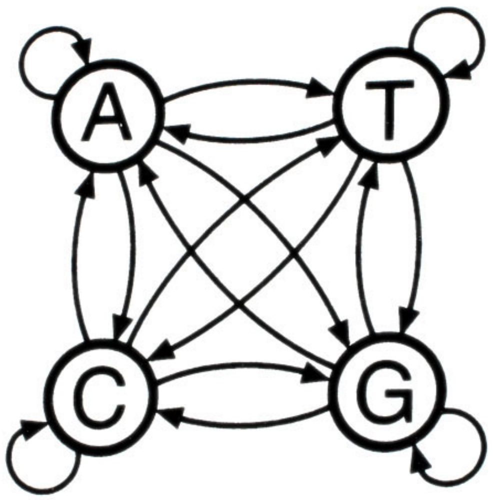

Figure 3. A hidden Markov model for DNA generation. A Markov chain consists of a number of generator states and their transition probabilities; in other words, the model encodes the features of a sequence and how these relate. [16]

Generally speaking we can't apply this method directly on the raw and noisy input we acquire from sensors, as it works best on slightly more high-level and de-noised features as frequency diagrams for speech recognition [15]. Once the model has been created it can be used to predict what generator was used for the current input stream; in the case of speech recognition this could work on all levels ranging from phoneme detection to full sentence construction. Though hidden Markov models can work very well, for good operation it is necessary to have a reliable set of such high-level and continuous features [57]. As mentioned earlier, in speech recognition tasks these are relatively easy to extract: frequency bandwidths, power, onsets, offsets and peaks are easy to detect and quite informative to the recognition models. However, this does not go for many other temporal pattern problems like the classification of signals from artificial whisker sensors. High levels of noise and over-saturated frequency bands do not allow such nice pre-processing and block identification and thus requires us to find recognition methods that can inherently deal with these difficulties.

2.3 Recurrent Neural Networks

Artificial neural networks are mathematical models that capture important elements of how we think real neurons operate. Besides being a very potent and easily applied technique for classification, they also offer high noise-robustness: they can be trained to provide stable and reliable output even when the input suffers from relatively high levels of noise. While feed-forward neural networks only have connections from one layer to the next, recurrent neural networks can also have 'backward' connections. This creates a dynamic flow of information in the network that allows for a form of internal short-term memory. Unlike the previously mentioned tapped-delay-line solution, these networks pre-process input and can be trained to internally store important features to base future decisions on [33]. However, exactly these internal temporal dynamics make it much harder to train the network to provide a specific answer for a series of inputs. Theoretically one single input characteristic from moments ago can radically change the current state of the network.

Back-Propagation Through Time (BPTT) [33] solves this in a rather extreme approach: gradually folding out the network for each input to store its network dynamics it creates one big feed-forward network. The well-known back-propagation algorithm [59] can train this resulting network as normal. Apart from the possible storage problems of folding out a complex recurrent network or dealing with long time series, this approach has two major drawbacks: it can only train in an epoch-based manner and suffers from fading error-signals. Though having a more real-time nature, Real-Time Recurrent Learning (RTRL) [65,66] does also suffer from this inability to recognize long-term relations between in and output. Error-signals are propagated to preceding connections and are likewise decreased according to their seeming importance. The longer these chains of connections grow, the lower the error signal becomes and the less effect it will have during training. Effectively this means that no long-term correlations can be learned [33].

While long-term is a relative judgement of course, when dealing with real-world data this often means high-resolution recordings of multiple seconds. Raw data from microphones needs to be sampled at least 2KHz to be able to extract frequencies informative for speech recognition. Long-Short Term Memory (LSTM) [30] was proposed as a solution able to recognize temporal patterns spanning both the short and this long-term range. Instead of using standard sigmoidal neurons it employs complex cells that are able to pick out informative features from the input stream. In order for this mechanism to work properly the input needs to be as clean as possible, noise can easily push it of its track.

Even though performance of these techniques is quite reasonable on their domains, there still remain a couple of intrinsic drawbacks when we want to apply them for recognition of patterns in noise rich and high-resolution streams. The main concerns are that training is very slow and needs lots of memory (BPTT), execution and analysis are not real-time (statistical methods) and/or reasonable performance is restricted to specific temporal problems (LSTM). Without extra adaptations these methods all suffer from noisy input, making them ill-suited for direct real-world applications [41,42].

2.4 Biology Does Do It Better

Besides the critique above, none of the mentioned methods are actually biologically very plausible [38,55]. While plausibility is not our ultimate goal on itself, we can't deny the fact that real neural structures are extremely good at real-time (temporal) pattern recognition [14,24]. For example, barn owls hunt at night and can locate possible prey in complete darkness, using solely their auditory system with a precision of 1-2 degrees of angle. This corresponds to the detection of the few (<5) microseconds delay in the moment of the patterns perceived by both ears [46], as well as judging whether this sound might come from a prey and is worth investigating. Experimental data shows that rats [13,56] can discriminate textures very precisely by brushing their whiskers against it and subsequently recognizing the vibrations from these specialized hairs [9]. They do this with pretty much the same precision as humans using their fingertips [9]. The rats, however, are much more specialized at this and do it using much simpler sensors and a smaller area of the brain. It is not known how they do this, there is seemingly no difference in the activity of the neurons involved when different textures are presented [54,56].

We simply have to admit that biology does do a much better job at temporal pattern recognition than we currently can do computationally. Artificial neural networks are very potent classifiers, but traditional models simplify the operation of biological neurons a great deal [46]. The main assumption that neurons encode their message in the average firing activity, for instance, cannot hold for the examples above: those neurons have to react non-linearly to changes in their input within milliseconds. Using neuron models that have a higher level of realism might not only provide us with computationally more powerful artificial neural networks, it could also provide insight in how real neural structures may operate [25].

Chapter 3
Spiking Neural Networks

Models of spiking neurons [63] try to capture the rich dynamics found in real neural structures with much greater detail than traditional neuron models did. Like real neurons, communication is based on individually timed pulses. Pulse coding, as this is called, is potentially much more powerful for encoding information. It is possible to multiplex much more information into a single stream of individual pulses than you can transmit using just the average firing rates (hence rate coding) of a neuron. We know, for example, that the auditory system combines amplitude and frequency very efficiently over one channel [46].

Another great advantage of pulse coding is speed: lots of it is gained as neurons can be made to react to single pulses, allowing for extremely fast binary calculation. The human brain, for example, can recognize faces in as little as 100ms. The involved neurons have to decide in (literally) split-second whether or not to fire. This would simply be impossible if they'd use rate coding, the averaging windows would become so small that no reliable output could be generated [62]. The majority decision of a group of neurons can be used to generate more reliable output: this is called population encoding. Though we find this in use in many parts of the brain, the massive amounts of neurons that would be needed to allow for such fast facial recognition are simply not present in the visual cortex [13,38].

Like real neurons, spiking neurons have an ever-changing internal state, the membrane voltage [38]. When a threshold value is crossed, an action potential is generated that traverses from the soma over the axons (see fig. 4). It is communicated to the dendrites of other neurons

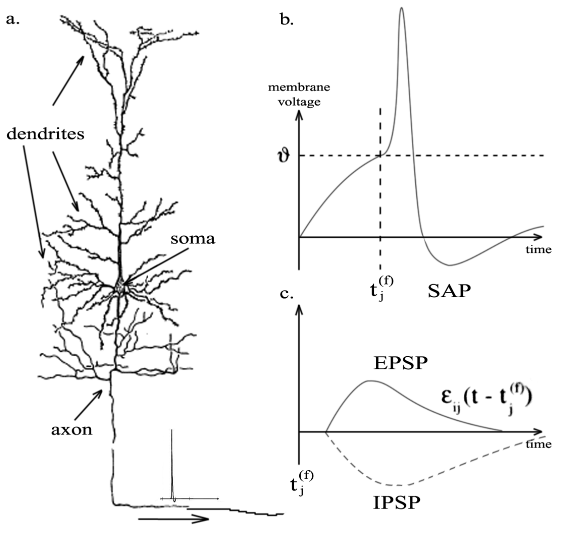

Figure 4. (a) Schematic drawing of a neuron. (b) Incoming postsynaptic potentials alter the membrane voltage so it crosses threshold value ϑ; the neuron spikes and goes into a refractory state. (c) Typical forms of excitatory and inhibitory postsynaptic potentials over time. [25]

by connections between these two called synapses. Such an action potential is a short (1ms) and sudden increase in voltage that we, due to their form and nature, refer to as a spike or pulse. Though all spikes look alike, their effect on connected neurons may be completely different. Connections between neurons are formed by synapses, complex signal transducers [38] that help the signal to cross the physical gap between the two neurons. An incoming spike triggers the release of neurotransmitter chemicals that cross the synaptic gap and generate a post-synaptic potential [25]. The type of synapse and amount of released neurotransmitter determine the strength and type of the post-synaptic potential: it may either heighten (excitatory, EPSP) or lower (inhibitory, IPSP) the membrane potential of the post-synaptic neuron. Real neurons use only one type of neurotransmitter in all of their outgoing synapses, effectively making the neuron itself either excitatory or inhibitory [38].

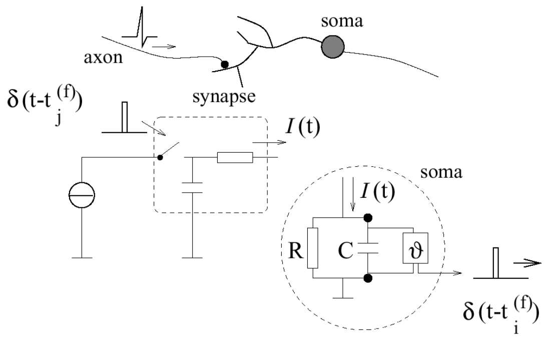

Figure 5. Schematic drawing of the integrate-and-fire neuron. On the left side, the low-pass filter that transforms a spike to a current pulse *I(t)* that charges the capacitor. On the right, the schematic version of the soma, which generates a spike when voltage *u* over the capacitor crosses threshold [25].

3.1 Pulse Coding Models

Many different models [25] of spiking neurons exist, of which the Hodgkin-Huxley model is by far the most detailed and complex. It was based on experiments with the large neurons found in squid. This set of differential equations realistically describes the exact reactions of the neuron's membrane potential to the influx of chemicals and electric stimulation. However, as this realism comes at a large computational cost it makes the model less suited for simulations of large networks. As our goal here is to find a model that is realistic enough to be successfully used in temporal pattern recognition we will only cover the much simpler integrate-and-fire (I&F) model. This model (see fig. 5) is an instance of the general threshold-fire model that describes that neurons fire if their membrane potential rises beyond its threshold value. It is very commonly used for networks of spiking neurons, as it is quite simple to understand and implement. However, it does approximate the very detailed Hodgkin-Huxley model relatively well and so captures generic properties of neural activity [25].

As we've seen in the previous chapter, action potentials are all very similar. We can therefore forget about form and characterise these by their firing times $t_i^{(f)}$. The lower index i indicates the neuron, the upper index f the number of the spike. We can then describe the spike-train of a neuron as

$$F_i = \{t^{(1)},...,t^{(n)}\} \tag{3.1}$$

The variable u_i is generally used to refer to the membrane potential, or internal state, of a neuron i. If a neuron's membrane potential crosses threshold value ϑ from below, it generates a spike. We add the time of this event to F_i, defining this set as

$$F_i = \{ t \mid u_i(t) = \vartheta \wedge u_i'(t) > 0 \} \tag{3.2}$$

When a real neuron generates an action potential the membrane potential suddenly increases, followed by a long lasting negative after-potential (see fig. 4b). This sharp rise above the threshold value makes it absolutely impossible for the neuron to generate another spike and is therefore named absolute refractoriness. Afterward, the period of negative spike after-potential (SAP) makes it less likely that the neuron fires again and is called the relative refractoriness.

3.2 Integrate-and-Fire Neurons

In the integrate-and-fire model spikes are seen as short current pulses that travel down the axons. Once it arrives at a synapse, the short pulse is transformed by a low-pass filter into a current pulse $I(t-t_j^{(f)})$ that charges the next integrate-and-fire circuit. This increase in voltage can be seen as the postsynaptic potential. Once the voltage over the capacitor goes above threshold value ϑ, the circuit shunts, sends out a pulse and resets itself. We write

$$\tau_m \frac{\partial u}{\partial t} = -u(t) + RI(t) \tag{3.3}$$

to describe the effects on membrane potential u over time, with τ_m being the membrane time constant in which voltage 'leaks' away. The neuron fires once u crosses threshold ϑ and a short pulse δ is generated. An important variable is the input resistance R of the circuit, typically set to values in the order of 1MΩ. To enforce a period of absolute refractoriness after firing, we force u to $K<0$ for a period of δ^{abs} during which the integration process is halted. Relative refractoriness can be added by setting the membrane potential afterwards to u^{reset} and restarting the integration process.

$$I_i(t) = \sum_{j \in \Gamma_i} c_{ij} \sum_{t_j^{(f)} \in F_j} \delta(t - t_j^{(f)}) \tag{3.4}$$

Because incoming pulses have a finite short length, a neuron's input current I will often be 0. Once a spike arrives, it is multiplied by synaptic efficacy factor c_{ij} to form the post-synaptic potential that charges the capacitor, for which we find all neurons j connected to i from collection Γ_i. This effect lasts a while longer than the duration of the pulse itself though, as the membrane potential only slowly leaks exponentially towards its resting potential.

Biological data shows that axons and dendrites are fast while synapses are relatively slow at pulse transmission. Inhibitory connections generally use faster neurotransmitters than excitatory synapses. We can introduce synaptic or axonal transmission delays Δ^{ij} in the pulse definition function δ itself, or simply shift the incoming spike window. By adding an extra term i^{ext} to equation 1.4 we can also model external influence to the neuron.

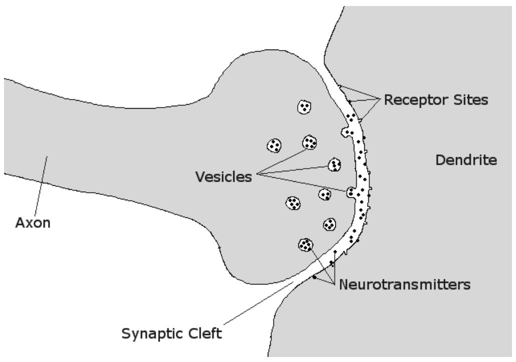

Figure 6. Schematic drawing of a synapse. When a pre-synaptic action potential arrives vesicles containing neurotransmitter chemicals may fuse with the membrane, delivering their content into the extra-cellular fluid of the synaptic cleft. When these chemicals bind to recep tors on the dendritic side of the synapse a post-synaptic potential will be generated, it's nature depending on the type of synapse and chemicals. [68]

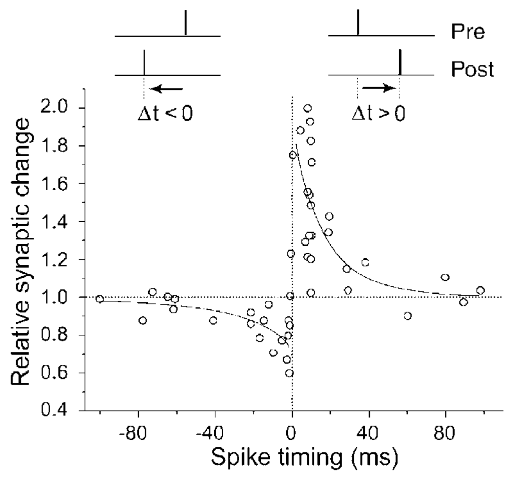

Figure 7. Typical asymmetric Spike-Timing Dependent Plasticity window: pre-synaptic action potentials that shortly closely before the post-synaptic neuron spikes cause LTP, while late arrival leads to LTD. The LTP time window is shorter, but has a higher peak. [6].

$$I_i(t) = i^{ext} + \sum_{j \in \Gamma_i} c_{ij} \sum_{t_j^{(f)} \in F_j} \delta(t - t_j^{(f)} - \Delta^{ij}) \tag{3.5}$$

As the above equations show, the integrate-and-fire model is conceptually and computationally relatively simple, though inherently much heavier than rate coding sigmoidal networks. An advantage of the model is that it is relatively easy to integrate it in hardware, allowing for very fast operation. It has a close relation to the more general spike-response model (SRM) and can be used like it by rewriting it into the correct kernels η and ε that form the heart elements of the SRM model [25].

3.3 Dynamic Synapses

Besides signal transducers, real synapses also are extremely complex information pre-processors that show strong dynamics at many different temporal scales [38]. In fact, they act as dynamic memory buffers, altering transmission behaviour depending on the form of the spike train it recently received [49]. Though most artificial neural networks use synapses that only act like static weighing functions, it has been shown that synapses that model more dynamic features also found in real synapses, like facilitation and depletion, are computationally much more powerful [47].

In biology, once an action potential arrives at a synapse (see fig. 6) there is a probability that a vesicle of neurotransmitters releases its contents into the extra-cellular fluid of the synaptic gap. These chemicals cross over to the post-synaptic side of the synapse, where they act on matching receptors on the membrane. Whether this postsynaptic potential is excitatory or inhibitory depends on the type of neurotransmitter that is released, its size on the amount of ion gates opened by receptors activated by the neurotransmitters. Afterwards the neurotransmitter chemicals are actively recovered into the pre-synaptic node, ready for the next round of duty [38]. When a spike arrives, the state of the synapse is very important in whether and in what degree a post-synaptic potential is caused. The amount of available neurotransmitter vesicles, amount of receptors and facilitation and depression recovery times all play important roles in defining the state of a synapse and thus how it will processes information encoded in the incoming spikes [49].

By using such dynamics instead of static weights in the model of the synapse, the network gains an extra degree of non-linear dynamics: allowing for more diverse and input specific dynamics displayed in the network. The mechanism operates on the neurotransmitter release probability, increasing the likelihood of spikes early in arriving spike-train to be transmitted [25]. Consequently, the amount of readily available vesicles for spikes that arrive later in the sequence is decreased. This leads to the effect that signals that arrive after a period of inactivity are much stronger and reliably transmitted, allowing for strong recurrent activity without network destabilization due to over-activation [49].

Using dynamic instead of static synapses increases the required computation significantly. Instead of just weighing input, each synapse has a dynamic state that changes over time and inputs. We can capture the dynamic state and information transmission of such synapses by the following set of recursive equations:

$$A_n = A \cdot u_n \cdot R_n \tag{3.6}$$

$$u_{n+1} = U + u_n(1-U)\exp(-\frac{\Delta t_n}{F}) \tag{3.7}$$

$$R_{n+1} = 1 + (R_n - R_n u_n - 1)\exp(-\frac{\Delta t_n}{D}) \tag{3.8}$$

$$u_1 = U, R_1 = 1 \tag{3.9}$$

The four main parameters to this model are *A, U, F* and *D*. Seen from the outside it is the synaptic efficacy A_n that alters over time when spikes arrive. Internally this effect is determined by the dynamic state variables u_n and R_n. These are related to the standard fraction of synaptic resources for spike transmission *U*. *F* and *D* are two time constants that describe time necessary for the synapse to recover from facilitation and depression [47].

The first spike to arrive is transferred with maximum efficacy, but it influences the fraction of synaptic resources R_n available to transmit the *n*th spike. The actual fraction of neurotransmitters used to transfer that spike is modelled in $R_n \cdot u_n$. In other words, if another spike arrives shortly after there will be less neurotransmitter chemicals readily available and it's efficacy will thus be lower. The second term of equation 3.8 describes how much chemicals are still missing at the time *t* since spike *n*. A depleted synapse is said to be 'depressed', as it does not hold enough available vesicles to evoke a proper post-synaptic potential. Depression factor *D* defines how fast the synapse can recover its chemicals from the synaptic cleft.

The fraction of the available resources *R* actually used for the (*n+1*)th spike depends on the synapse's degree of facilitation, or how soon after a spike large parts of the resources will be used again. For different combinations of values for factors *D* and *F* totally different synaptic information pre-processing and transmission behaviours can be achieved. Inhibitory synapses are generally much faster than excitatory ones, but show relatively slow facilitation and recovery from depression.

Recent work has shown that many synapses are likely to be equipped with two types of neurotransmitters: slow releasing and fast recovering versus fast releasing and slowly recovering messenger chemicals [53]. This has not yet been incorporated in the above model, but may lead to even more dynamic and realistic synaptic signal transmission behaviour.

3.4 Training

Though networks that employ pulse coding theoretically provide a large computational power, we first need to harness or use the dynamics of these networks before we are able to use them for successful application.

3.4.1 Supervised Learning

For a long time, supervised training methods for spiking neural networks have been lacking. In more traditional models of artificial neurons, training techniques could rely on the predictability of a neuron's output given a certain input. Spiking neurons however, behave in a much more dynamic manner: firing with either fixed frequency, short bursts or not at all and are able to habituate to input [38,46]. Existing techniques like backprop [59] therefore can't be applied directly: the network's reaction is determined more strongly by the exact timings of spikes than the average activity of neurons. Spike-prop, an adaptation [7] of this powerful algorithm, can train a network of spiking neurons to react to input with single individually timed spikes. Its applicability is limited however, as in order to work it requires an overly complex network of many individually specifically delayed and differently weighted synapses per connection between two neurons. These requirements increase the computational load for networks tremendously. Combined with the fact that the resulting networks are quite sensitive to noise, it can only be put to effective use in specific domains [7].

3.4.2 Unsupervised Learning

A variety of methods for unsupervised training of spiking neural networks have been around for a while, though [39]. Spike-Timing Dependent Plasticity (STDP) [1,61] is a powerful form of Hebbian learning [29] that delicately alters synaptic efficacy (weight) according to the precise timing of post and pre-synaptic activity. Synaptic modification only occurs if the difference between pre and post-synaptic firing is small. The sign of this difference determines whether the connection is strengthened or weakened. By using such an asymmetric window (see fig. 7), STDP can stabilize network activity as it puts neurons in temporally sensitive states and introduces competition between neurons [6].

Synapses influenced by this form of spike-timing dependent synaptic plasticity [61] are forced to compete for control of the timing of postsynaptic action potentials. Connections that receive spikes just before a postsynaptic spike is triggered are strengthened, as STDP causes Long-Term Potentiation (LTP) at these synapses. Analogously, it weakens those that fire too late by invoking Long-Term Depression (LTD). This way, STDP modifies (see fig. 7) excitatory synaptic strengths until there exists a sufficiently high, but not excessive, probability of a presynaptic action potential being shortly followed by a postsynaptic spike. STDP can be extended such that it also modifies the short-term synaptic plasticity properties of a synapse, an effect that is called synaptic redistribution [1] and works best in combination with dynamic synapses.

The element of competition, punishing late firing, provides a powerful method to adapt individual neurons to listen most to those inputs that are consistent at predicting strong post-synaptic responses. This leads to two effects in the network: groups of neurons are encouraged to fire in correlation and neural pathways in the network with low latency are formed and strengthened as synapses of longer-latency or less-effective inputs are weakened [40]. Basically, depending on the input pattern the network will be able to react with fast correlated activity.

Even though these techniques can regulate network activity, they remain methods for unsupervised training and can therefore not be used to steer the network's response towards certain input. As an alternative to supervised learning, evolution has been successfully employed to evolve networks of spiking neurons for various tasks of robot control [23,43] and temporal pattern recognition [41]. The main difficulty in applying evolution is the definition of the fitness function; evolution does not strive to find the perfect solution, but settles for an economic one. The fitness function should enable evolution to find the best solution by creating a smooth fitness space. This is virtually impossible for tasks with relatively long-term correlations. Especially for tasks of temporal pattern recognition this often resulted in premature convergence to a local maximum [42].

3.5 Using spiking neurons

Spiking neural networks have many nice properties that set them apart from the more traditional networks of rate-coding neurons [25]. They have an inherent notion of time that makes them seemingly particularly suited for processing temporal input data [31,63]. Their non-linear reaction to input provides them with strong computational qualities, theoretically requiring just small networks for complex tasks [41]. Changes in synaptic efficacy found in living neurons have been formalized into unsupervised learning methods that successfully can cluster network activity for input classes [39,61]. All these features combined capture a lot of detail of our current understanding of how real neurons operate.

However, what we currently do not understand yet immediately forms the Achilles heel of applying spiking neural networks: supervised learning. Controlling the response of these highly non-linear and time-sensitive networks has been proven to be very difficult. The algorithms we have found can only be applied on very limited domains. It is very unlikely that real neural structures just rely on unsupervised Hebbian learning for training of tasks, but we do not understand how they do organize and adapt to do their tasks.

Novel insights suggest that there might not be such a thing as a neural code that we need to understand and control, but that the power of spiking neural computation lies in making use of their dynamics [34,48]. This rather radical change in approach thus leaves the idea that we need to steer and control the network dynamics behind and instead proposes that we rather try to make use of those dynamics by for instance recognizing state trajectories in the network. Spiking neural networks are so brought right back in the realm of applicability, as even in noise rich environments their dynamics have been shown to be reliable [62].

Chapter 4
Liquid State Machines

Independent of each other, Maass [48,50] and Jaeger [34,35] developed the similar concepts of respectively the Liquid and Echo State Machines. These similar mathematical frameworks claim to allow for universal real-time computation without stable states or attractors on continuous input streams. Both operate by feeding a continuous input stream into a sufficiently complex recurrent neural network. Due to the intrinsic dynamics in the network it is likely to contain substantial information about all recent inputs. What this network, which we call the excitable medium, does is transforming the low dimensional temporal input into a higher dimensional 'liquid', or 'echo' state. Though this dynamic state might be unintelligible for humans, we can successfully apply spatial classification or pattern recognition techniques in readout units that can extract valuable information about recent inputs from it.

The above is rather abstract and best explained by the following example: imagine a rock and a pond and throw the rock into the water. In fact, here the rock is a low-dimensional temporal input: the rock and throw have some properties but these are only expressed very briefly. The resulting splash and ripples that are created can be seen as the reaction, or 'liquid' state. These ripples propagate over the water's surface for a while and will interact with the ripples caused by other recent events. The water can thus be said to retain and integrate information about recent events, so if we're somehow able to 'read' the water's surface we can extract information about what has been recently going on in this pond. We refer to this trained spectator as a readout unit that we can ask at any one time what's going on in the pond, provided that we can show him a picture of the water's surface.

4.1 Formal Definition

The concepts of the Liquid State Machine (LSM) [48] and the Echo State Machine (ESM) [34] are very similar and only differ in detail. The ESM setup has been mainly applied using traditional recurrent neural networks, while LSMs have been put to work primarily using spiking neural networks as the 'liquid medium'. As we strive to investigate biologically more probable temporal pattern recognition methods we are more interested in the latter setup. We will therefore only use the name 'liquid state machine' and its abbreviation, but mainly for not having to use both names all the time. A mathematical framework formalizes the intuitive notion that infinitely large and sufficiently dynamic and complex systems as the liquid can be guaranteed to provide universal computational power in the LSM, like Turing machines [48]. While Turing machines do this for off-line computation on discrete input, the LSM achieves real-time computation on analog functions in continuous time, without stable states.

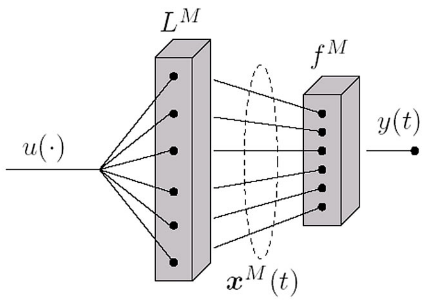

Figure 8. Schematic principle of the liquid state machine. A continuous stream of input values *u(·)* is fed into 'liquid medium' L^M; a sufficiently complex excitable medium of which outgoing signals $x^M(t)$ define the liquid state. These values are propagated to read-out unit f^M that generates output *y* for the current time step *t* [48].

Formally, the LSM (see fig. 8) works as follows: A continuous input stream *u(·)* of disturbances is injected into excitable medium L^M that acts as a liquid filter. This liquid can be virtually anything from which we can read it's current liquid state $x^M(t)$ at each time step *t*. This liquid state is mapped to target output function *y(·)* by means of a memory-less readout function f^M · Examples of media that have been successfully used as the liquid include recurrent networks of sigmoidal [34] neurons, temporally dynamic and biologically realistic spiking neurons [48] as well as ordinary real wet water [22].

4.1.1 Infinite State Machines

As the names of both the LSM and ESM hint, there is a resemblance with finite state machines. However, in the finite state machine the full state space and transitions are carefully constructed and clearly defined; quite unlike the rich dynamics found in the medium of an LSM. These state machines could be seen as a universal finite state machine, as a sufficiently highly dynamic 'liquid' can implicitly contain the states and transitions of many complex finite state machines. The large advantage of the LSM is that we do not have to define the states themselves, as the readout units can be trained to extract the state from the continuous analog state of the liquid. We only have to make sure that the reaction of the liquid is reliable in the sense that it reaches identifiable states after a predefined number of time steps of input.

The dynamics in the medium are of great importance, as it allows for information to be temporally stored and integrated. It may therefore seem surprising that randomly generated and connected networks with fixed weights perform really well, handcrafting is said to be unnecessary. Exact performance differs per generated 'column' of neurons, but their computational power is reported to be quite high. Size and level of connectivity of a column are important to its memory span, but even more so are the dynamics of both neurons and synapses.

4.1.2 Separation Property

The most important aspect of the liquid is to react differently enough to different input sequences. The amount of distance created between those is called the separation property (SP) of the liquid. The SP (see fig. 9) reflects the ability of the liquid to create different trajectories of internal states for each of the input classes. The ability of the readout units to distinguish these trajectories, generalize and relate them to the desired outputs is called the approximation property (AP). This property depends on the adaptability of the chosen readout units, whereas the SP is based directly on the liquid's complexity.

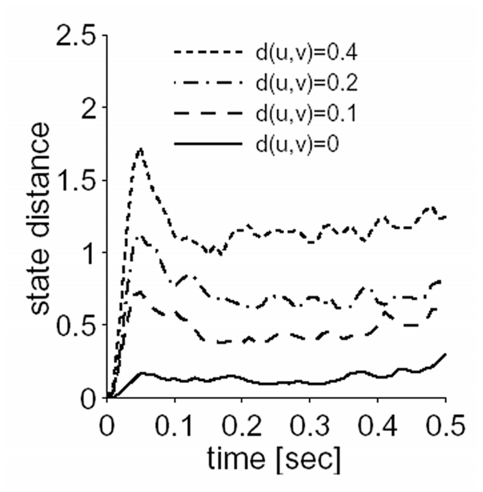

Figure 9. The separation property of a generic neural microcircuit, a randomly generated LSM. Plotted on the y-axis is the distance between the two states of the liquid in reaction to input signals that have Euclidian distance *d(u,v)* [48].

Jaeger [34] found that he could greatly increase the performance of an Echo State Machine by using recurrent networks of continuous leaky-integrator neurons instead of standard recurrent sigmoidal networks. The main difference between these two types of synaptic connections is the ability of the former to react more non-linearly to temporal input and as such they strongly increase the non-linear dynamics of the ESM. Maass [50] found very similar results when he compared the performance of LSMs using liquids of spiking neural networks with either static weighing connections or the more realistically dynamic synapses. As such synapses allow for depletion and habituation they provide for more non-linear interaction in the network. The finding that more dynamic connections increases the computational power of the model is not surprising if we realise that using these neurons and synapses greatly enhances the temporal dynamics of the medium, making input trajectories more likely to be unique and thus allow for easier recognition of different input streams. In relation to the notion of the separation property, we can say that there seems to be a strong correlation between the computational power of the full system and the non-linearity of the liquid networks [51].

4.1.3 Liquid States

Identifying the difference between two states of the liquid is actually quite difficult [35,48]. Liquid states depend on the input streams they've been presented, preferably we would like to have a correlation between the similarities of the input streams to the resulting liquid states at fixed readout moments. To break the news, no ultimate distance measure between such liquid states exists, nor for temporal input streams. Due to the highly dynamic non-linear and temporal nature of the states, we intuitively see that simple 1-on-1 matching between two states will not work: even slight jitter in the two input patterns might cause states to seem different, while they're actually very alike. Still, it does provide some indication of the distance between the liquid's trajectories for two input streams. We can, for instance, use the average Euclidean norm of the simple distance between two network states over all recorded time steps. Formally, for comparison of two states we write:

$$d_s(t,u,v) = \left\| X_u{}^M(t) - X_v{}^M(t) \right\| \tag{4.1}$$

where $X_u{}^M(t)$ and $X_v{}^M(t)$ denote the liquid states at time t of input streams u and v. We can acquire a similar indication on the distance between two spike trains if we apply a low-pass filter, after which the resulting continuous streams can be directly compared. Of course this offers no ultimate comparison, as it does not take the neuron's dynamics and reactions into account.

4.1.4 Links to Biology

In the brain, the continuous sensory input streams are processed in neural microcircuits in the cerebral cortex [14,38]. These columns of neurons are anatomically and physiologically very similar and each of them performs many complex information integration tasks in parallel. Across the human brain we find more structures that are very similar to these microcircuits. In fact, we do not find these just in human brains but also in the brains of other species. This led to the thought that such heterogeneous columns of neurons are task independent and may therefore have potentially universal computational abilities on continuous streams of input [36]. This is a new insight in the understanding of neural computation that has some strong and important parallels with LSMs.

Edelman [17] pointed out a while ago that repetitive neural structures with universal properties are likely to have formed during the evolution of the brain. He proposed to investigate the computational power of such structures, but did not provide a framework like the Liquid or Echo State Machines. His research remained on the more philosophical grounds on how and why repetitive structures must've been used by evolution to create many of the talents the human brain has today.

4.2 Generating and Using LSMs

By using biologically sufficiently realistic neural networks as micro-columns in the liquid we might both be able to tap the power of the cortical microcircuits, while at the same time learning a fair bit about the capabilities of their biological counterparts. Maass [48] used randomly generated networks of 135 spiking neurons per column and used biological data to define parameters like connectivity, synaptic efficacy and membrane time constants. He used spike trains as input and those are fed to a random 20% of the neurons in the network. It remains a question how the level of realism relates to the computational abilities of these microcircuits.

Recapitulating, the basic action in the liquid state machine is the transformation and integration of continuous low-dimensional input streams into perturbations of the higher dimensional liquid to facilitate feature extraction [58]. The internal dynamics ensure that the current state of the liquid will hold information about recent inputs. The high-dimensionality of this liquid state helps making recognition much easier. First of all, during the memory-span of the liquid the complex low-dimensional temporal problem is turned into a more uid the complex low-dimensional temporal problem is turned into a more easily solvable spatial recognition task. Secondly, the high dimensionality and dynamics make it theoretically virtually impossible that two different inputs lead to exactly the same network state or state trajectory. In fact, inputs that are radically different are likely to lead to vastly different states of the liquid: allowing the readout units much easier discrimination.

4.2.1 Readout Units

Perceptrons, or threshold gates, have been around for over 40 years. Although they allow universal Boolean computation and are universal approximators for continuous functions, initial enthusiasm faded as no method for solving non-linear separable problems with satisfactory performance was discovered. Perceptrons compare the summed input received from weighing synapses to a fixed threshold: their output is high if the input is above, or low if it is below this value. They can, however be put to very good use as simple readout units in LSMs. After training with linear regression they can both be applied for classification and for the continuous extraction of information from the liquid state. The presentation of the liquid state to these units is of utmost importance however, especially if both mechanisms speak different 'languages': i.e. use pulse and rate coding mechanisms. Though rather simplistic, a low-pass filter can be applied to transform the spike-trains into continuous output that can be weighted and fed to the readout perceptrons. [48].

However, there are limitations to the power of linear regression and single perceptrons. Information in the liquid can be encoded in non-linearly separable ways and can thus theoretically not be extracted by these simple readout units. Computationally stronger is the use of perceptrons in pools, using the average activation of all perceptrons as output, or the committee machine. In the latter the Boolean output depends on the majority of the votes in the pool. Still, training these units is complex. Mechanisms that can solve non-linear separable problems in dependable manner are multi-layer perceptrons trained using backprop [59]. These have been successfully applied to extract input timings from low-passed liquid states [26].

4.2.2 Parallel Delta Rule

Recently, Auer [2] introduced the p-delta rule for parallel perceptrons, neural networks of a single layer of perceptrons. Its performance is comparable to that of other state-of-the-art classification algorithms, but is much less complex: only one layer of perceptrons and just one global 2-bit communication signal is required. In contrast, the backprop [59] algorithm for multilayer perceptrons needs to communicate high-precision floating-point values between neurons, making it much less biological plausible and computationally more taxing.

The largest difference with standard linear regression is that we'll have to adjust the full weight vector of the perceptron pool to change the output towards the desired level. There are a number of different methods thinkable, but simply adjusting all weights is claimed to work for the parallel delta rule. Though harsh, it ensures that at least the right weights are adjusted and change occurs. The weight adjustments can be calculated in both batch and iterative manners, from which we present the latter here that can easily be translated to batch updating. We calculate the amount of weight change for each connection by

$$\alpha_i \leftarrow \alpha_i - \eta(\|\alpha_i\|^2 - 1)\alpha_i + \eta \begin{cases} -z & \hat{o} > o + \varepsilon, \alpha_i.z \geq 0 \\ z & \hat{o} < o - \varepsilon, \alpha_i.z < 0 \\ \mu z & \hat{o} \leq o + \varepsilon, 0 \leq \alpha_i.z < \gamma \\ -\mu z & \hat{o} \geq o - \varepsilon, -\gamma < \alpha_i.z < 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

where $\hat{o}$ is the output of the pool, α is the weight vector, z the input vector , η the learn rate and o the target output. The particular choice of clear margin parameter $\gamma>0$ is important to ensure good performance of the algorithm. It helps in tuning the weight vectors of which the dot product is already of the right sign, steering the vector output away from 0. Parameter μ is used to scale the clear margin importance, but can be left to 1 as it's claimed to be of insignificant importance. The error function [2] to be used with the p-delta rule is rather complex, but tends to scale the error exponentially with the distance between the actual and desired output.

Though it is hard for us to judge the level of difference between two liquid states, input patterns do generate identifiable state trajectories. The high dimensional liquid state thus allows for easy pattern classification, even for memoryless linear readout units like perceptrons [5]. The training of linear readouts is quite easy and even very robust; due to the high liquid dynamics it is very unlikely they'll end up in local minima. This means that relatively simple and scalable training methods can be employed for use in the LSM. The p-Delta rule [2] also fits this description, as arbitrarily sized groups of perceptrons can be used as a readout unit, while training does not become overly complex. Therewith it makes the approximation property of such readout units scalable to the required level. Recently it has been shown that a readout unit consisting of just single neurons [28] can achieve roughly the same classification power as more sophisticated units like pools of perceptrons trained using the p-delta rule. Exact results have not been published, though, so it is yet to be seen how well single or pairs of (spiking) neurons can be employed for extracting information from the liquid state.

4.3 Applications

The concept of liquid state machines seems to be very powerful and there don't appear to be any strong a priori limitations to their computational power. Using sufficiently large and dynamic generated networks of spiking neural microcircuits as the liquid allows for somewhat biologically realistic real-time computing with fading memory. Their performance on a range of benchmark applications [48,50,51] seems to be quite in line with the expectations the model raises.

One of the most well-known temporal benchmark tasks in literature is the Hopfield & Brody speech recognition task where recordings of spoken numbers zero to nine have to be correctly classified [31,32]. The dataset consisted of 10 recordings each of 5 female speakers, totalling to a set of 500 inputs. Each of these recordings was encoded in 40 individual spike trains. Their original solution dubbed 'mus silicium' or simply 'organism' had the rather unrealistic requirement of a maximum of one spike per input line. The performance of this solution was not surpassed in an Internet competition.

However, the average squared error after training of an LSM with a liquid of 135 integrate-and-fire neurons was lower than that of the much larger and more complicated solution by Hopfield & Brody [31,32]. With noise added, the difference in performance increased even more. The largest and most important difference between the two approaches was the real-time computation in the LSM. It classified the input streams directly after they ended and did not need 400ms for evaluation of the generated levels of synchrony [48]. A complex blended combination of the LSM and spiking synchrony approaches was found to perform with reasonable performance as well [36].

The real-time parallel computation ability was explored in a task where various features of spike trains had to be extracted by different readout units at the same time. These features included, amongst others, the summation of spiking rates over different time windows and the detection of both a spatio-temporal pattern and spike activity coincidence. Though no exact performance measure was provided, it was both stated and shown in example that the LSM response was very near to the desired outputs [50].

Full spatial-temporal problems are of much higher complexity than the tasks described before, as the system will have to deal with correlations between different sensory input streams. The applicability of LSMs to such problems was explored by presenting it a simulated visual input grid in which an object (ball or bar) moved linearly. 36 readout units were trained to predict the sensory activation by the object moving through the grid for 25ms and another such group for 50ms. Predicting ball movements is a complex task. It requires integration of information from past and present inputs about location, speed and object class. The most interesting part of the experiment was the unsupervised training of 28 readout units to predict where the object was going to exit the sensor field. The average error on this task was about 4.9 degrees of angle of movement of the object [50]. As there was no investigation to the real complexity of the task, it might be that these results could be achieved by relatively simple integration of local activity in the network. Though it was shown that LSMs could be used for non-linear control, there was no real analysis about the true complexity of the task of moving the simulated robotic arm to designated positions [37].

The results of published experiments suggest that the LSM model with spiking neural networks as a liquid is generally powerful. However, no data is available yet on the performance of tasks with more direct and noisy input from the real world. Nor have there been any experiments specifically exploring their applicability on temporal pattern recognition, let alone for patterns from the real world.

Chapter 5
Approach

Even though all reports published on Liquid State Machines are very positive, a lot of big questions about the possibilities and applicability of the model remain unanswered. For starters, there has been no thorough investigation to what type of network integrates information best. Even though spiking neurons are more biologically realistic than rate-coding sigmoidal neurons it is open to debate whether they effectively add computational power [25]. We will investigate in what configuration integrate-and-fire neurons provide the best pre-processing for temporal pattern recognition.

The main claim that a more dynamic liquid relates to higher computational power seems to be backed by the experiments comparing dynamic and static synapses. However, those results have been acquired on very limited experiments and may not easily scale up for general LSM purposes [50]. Also, only relatively large networks have been tested as liquids for a small range of tasks. There's no clarity on whether performance always scales up with column size, nor what network size can be expected to provide good performance for certain types of tasks [5]. We will compare the performance of static and dynamic synapses for all experiments we run.

5.1 Noise Sensitivity

Having a low sensitivity to noise is important for any task that operates either in, or with data from the real world. Apart from the speech-recognition task where inputs were jittered and performance improved [48], no results have been published yet on what effect noise has on general LSM performance.

Even so, the results acquired by those who took 'liquid' in LSM literally indicate a high noise resistance [22]. Their setup consisted of a glass bucket of water on top of an over-head projector and a web cam filming the interference patterns created by vibrating the water's surface with motors. The filmed result was then fed into a perceptron for classification.

Despite the extremely noisy nature of this setup they achieved to solve not only XOR, but could also discriminate ones and zeros from the Hopfield-Brody [31,32] task. The fact that no effort to reduce the level of noise in the system was necessary indicates that either Liquid State Machines can operate well under noise or the tasks were simpler than we assumed. However, results of experiments specifically addressing noise resistance have not been published.

5.2 Pattern Recognition

All published experiments using Liquid State Machines for temporal pattern recognition required the most simple form of recognition: classification of and between fixed patterns with little or no noise added to the liquid or input. While this still would be a daunting task for many of the recognition techniques covered in chapter 2, it is by no means realistic [14]. Classification of patterns generated from templates is much more complex and if successful, would be easy to apply in real world applications. Before this can be investigated however, it needs to be clarified whether the spiking neural networks used as a liquid do indeed provide short-term memory and whether exact spike timings in the input are really important.

5.3 Plan of Approach

We will try to address these questions in our experiments, hopefully shedding more light on how well Liquid State Machines with spiking neural networks as the liquid can be made to operate in temporal pattern recognition tasks. Because noise resistance is so important in operating in the real world we will give this extra attention in our experiments. We can achieve this by adding jitter and using a template based input generation: instead of allowing our LSM to recognize the relatively simple stable end-states we will force it to pick out stable features in the liquid's dynamics. By doing this performance is expected to be lower, but also much more stable. A more direct approach will be our investigation of the performance of an LSM on tasks where recorded inputs from real sensors are to be classified. These already have an inherent level of noise that will have to be overcome for usable classification or information extraction.

5.4 Liquid Stabilization

It is claimed that the high liquid dynamics ensure longer short-term memory and a better separation property. Basically the idea we set forward in this chapter is that the more dynamic and non-linear neurons and synapses can react to input, the better. It may therefore seem awkward to propose stabilization of the liquid dynamics as a possible desirable effect, but we claim that it could possibly lead to a better separation property at lower computational costs.

Though STDP is a destabilizing factor at individual synaptic level [39], at a larger scale it actually regulates network homeostasis [1], something highly desirable. By applying STDP to a liquid of spiking neurons, we can let the network adapt its activity to the inputs it receives [61]. Experiments will have to show what effects this has on the separation property and overall performance. It may possibly lead to easier recognition of activity patterns for different classes of inputs. STDP might lead to more stable invariant features as the network could adapt to classes of input streams, possibly leading to more powerful temporal pattern classification by the readout units. Experiments will also have to show whether this might reduce the need for dynamic synapses in the network.

5.5 Implementation Issues

Apart from the suggestions and questions raised above, there are some other specific issues that need to be addressed. One of the most important ones is speed. Though spiking neurons can relatively easily be implemented in hardware, simulation of this kind of neurons requires much more computational power than traditional artificial neurons do [46]. Though real-time pattern classification is the primary goal, especially during training and evaluation high simulation speeds are highly preferred. This leaves the big question how fast such large, dynamic and parallel networks can be simulated on standard sequential computers.

CSIM, the open-source LSM implementation by Natschläger [52] has a number of serious drawbacks. Although being optimised and written in C++, simulation and training are rather slow, which can only partly be explained by its strong use of the Matlab environment for visualisation. Other limitations are the lack of freedom for alternating input presentation and most importantly general parameter adjustments. We have therefore developed our own experiment framework that allows for easy and fast parameter space exploration, extensive logging and an easy to use graphical user interface. During the development of this software we tried to keep the differences between the network details as limited as possible to enable (future) comparisons between results and software performance.

Opposed to the spike-event-based simulation engine Natschläger chose to implement, we decided to use an iterative engine for our simulator. Event based simulators are theoretically faster, especially when network activity is sparse. However, our current full-traversal network operates at much higher speeds than CSIM. Typical simulations of liquids of 150 integrate-and-fire neurons run over 30 times faster than real time at 1ms resolutions on a typical workstation. Large networks of up to 2500 neurons with relatively high connectivity can still be simulated and trained at real time speeds, unlike what we could using CSIM. Apart from operational speed, the iterative nature of the network simulator also allowed us to enable extensive logging and visualization without taking a large performance hit.

Chapter 6
Methods

In order to investigate the usefulness of Liquid State Machines for the recognition of temporal patterns, we set up a series of experiments that should shed light on particular properties of the whole mechanism. In this chapter we will provide the reader with information about our general approach for setting up LSMs and running experiments.

6.1 Generating Liquid State Machines

Each network of integrate-and-fire neurons we used as a liquid was randomly generated, as were the connection weights from the liquid's low-pass output filter to the readout perceptron(s). For each experiment reported about in the next chapter we tried to find the settings that would allow for optimal classification. Initially we started our parameter-exploration in the vicinity of the parameter ranges Maass used in his principal and later LSM papers [48,51].

The neurons of the liquid pre-processor were assumed to be located on the integer positions of a 3-dimensional grid. These locations were only used during the build-up of the connectivity of the network, as connections were stochastically determined with the Euclidean distance as a parameter. We used a λ-function (see equation 5.1) to calculate connection probabilities between pairs of neurons. This method allows for controlling both the average number and average length of synaptic connections with a single parameter λ. For each pair of neurons in the network a uniform random process determined whether an actual connection was formed, self-recurrent connections were not allowed.

$$p_c(a,b) = C(a,b) \cdot e^{-\frac{D(a,b)^2}{\lambda^2}} \tag{5.1}$$

where we used the values depicted in table X for the connection-dependent scaling factor C, D is the Euclidean distance between the two neurons. The networks generated with this rule have a sparse and primarily local connectivity. A random ten percent of the neurons were chosen to be inhibitory. While Maass wanted to use values close to empirically found biological data, we were mainly interested in its effect on the computational power of LSM setup. Initial experiments showed that lower values for λ reduced chaotic effects while improving performance. We therefore opted for values around 1.2, instead of 2.0, for the key parameter λ. We will cover these findings in the next chapter.

Instead of using sensor neurons within the liquid, we provided a random ten percent of the neurons with sensory synaptic connections. These connections receive direct spike trains from our input generators and can be seen as coming from a neuron of which the input streams directly encode its activity. The synaptic efficacy of the connections was uniformly chosen within a range of [0,s_s] for sensory connections and [0,s_n] for normal interconnectivity. We will provide the exact settings used for each experiment in the next chapter.

All neurons had membrane time constants of 30ms, an absolute refractory period of 3ms for excitatory and 2ms for inhibitory neurons. The important membrane potential values were set to 0mV for reset and resting voltage and 15mV for the firing threshold. All neurons received a non-specific background current drawn from a normal distribution with a mean of 13.5nA and a standard deviation of 1nA. The input resistance was set to 1MΩ for all neurons. All synaptic delays were set to 1ms, irrespective of neuron types and grid distance.

C(a,b)	Neuron b Inhibitory	 Excitatory
Neuron a		
Inhibitory	0.1	0.4
Excitatory	0.2	0.3

Table 1. Neuron-type dependent scaling values for determining network connectivity. General trend is to generate few connections to inhibitory neurons, intuitively allowing those more feature specific reactions. [48].

6.2 Simulation Overview

Each experiment we ran consisted of a typically 50 individual runs. For each such run we generated a random liquid and appropriate readout unit(s). A run consists of 3 phases: initialisation, training and testing. During these three phases spike trains are being fed to sensory synapses of the networks. These input streams are either fixed or generated on the fly from a probability template. At fixed intervals (typically 200ms) a low-pass filtered snapshot of the liquid state is presented to the readout units. For the classifying perceptrons this input is weighted and if above 0.0 the output is a high (1.0) signal, otherwise it is low (0.0). For continuous extraction we used simple linear perceptrons of which the output is simply the weighted input. We also randomly choose the next input stream generator at fixed intervals, generally of equal length.

The output from the readout units is recorded at the last time step of each input interval. During the training phase either linear regression of the p-delta rule is applied to update the connections from the liquid to the readout unit. For classification the *n*th readout unit is supposed to only provide high output when the pattern of the *n*th class was presented. Classification performance is measured for each phase individually in percentage of correct identification. We only report the performance of the testing phase.

6.3 Input

A standard challenge in any data processing task is proper input and output transformation. Raw input data, i.e. the digitised data we acquire from sensors, seldomly fits what the data processor understands, so often we are forced to translate it into signals it does. The output we receive after processing can pose a similar problem if it's not easily clear what the data processor is telling us. Spiking neurons use trains of individually timed spikes to transmit signals [63]. If we want to use such a network, we'll have to make sure that we are able to properly translate our raw input into such spike trains. We also have to be prepared to receive and handle these temporal signals as the network's output.

The great computational power of spiking neurons lies in the fact that they react non-linearly to input; it is not so much the (average) number of incoming spikes that matters, but their exact timing that has a large effect on the output of the neuron [25]. This posed a problem that people have been struggling with ever since. The case is that we can quite easily devise methods to transform input in the average activity of spike trains, but nobody has been able to find a general rule that allows good use of exact timings of spikes [10]. Whenever we want to interface spiking neural networks with input other than spikes (which is usually the case) we'll thus have to find a good method to translate the temporal aspects of the input into spike trains [11]. As said, this is an art on itself and extremely important with respect to the theoretically achievable performance [46].

To be able to analyse the performance of our LSM setup we chose to generate single Poisson spike trains using probability templates as input. We can control and check the characteristics of these spike trains and can determine much about what features can and possibly are being used by the LSM [12]. To allow for such analysis of what our LSM set-up is capable of, we will keep our set of experiments limited to tasks with individual input streams that are switched over time. All sensory synaptic connections receive direct input from this single input stream. While such single input streams are highly unrealistic from a biological point of view [14], the good thing is that it allows analysis on the complexity of the task. In contrast, this is nearly impossible for the Hopfield-Brody speech recognition task [31], or actually nearly any published LSM application. As our approach for transforming the rough sensor data acquired from artificial whisker sensors into identifiable probability templates is novel, we will describe it in detail in the results section of this thesis.

6.4 Output

Decoding the output of the network poses us with similar encoding problems: because of activity sparseness we cannot simply use the binary output for our linear readout units. At many moments in time no spikes will be fired at all and with no network activity the readout units have no cue to base their output on [48]. Due to their high non-linearity, the membrane potentials of the spiking neurons neither provide a good indication for determining the readouts [11]. Instead, we apply a low-pass filter on the output of the liquid neurons with a window that reflects their membrane time constant of 30ms. By doing this we do throw away a lot of temporal information, but as the experiments show it suffices for our need [51]. It does rightfully raise the question why to use spiking networks at all: it may seem like a computationally intensive variant of a standard rate-coding network. However, the network's internal non-linear dynamics do rely on the exact timings. Using a proper temporal output decoding could possibly enhance the computational power of the mechanism [5,37].

During a fixed number of time steps input from one source gets fed to the sensor synapses of the liquid. Depending on the task, the liquid state is either extracted continuously at each step of simulated time or only at the last time step of an input period. We did not collect a series of liquid states to optimise our readout units with, but chose to train the LSMs on the fly; allowing the readout units to encounter a much more diverse range of inputs and so lowering the chance of over fitting.

The weights of connections to the readout units, initially uniformly chosen in the domain of [–0.05,0.05], were updated at fixed training intervals using standard linear regression or the p-delta rule. We tested both these algorithms for all tasks, and will report on our findings in the last section of the results chapter. These intervals were typically of the same length as the input streams. After each input period the new generator was randomly chosen. We could choose to reset the liquid state at these moments by erasing all spikes in the network and setting the membrane potentials to uniformly chosen random values between their resting and threshold value. While others [22,26,37,48,49,50,58] choose to do this at every input switch, we primarily investigated the more realistic situation where this does not happen. Residual noise from previous inputs will thus be present in the network. This not only makes the initial network state noisier, but the full setup more realistic for continuous online usage.

We used three distinct periods during each simulation, for initialisation of network activity, training and testing of performance. Apart from network activation, the initialisation phase allows STDP [61] to be applied before the actual training phase. Intuitively, changing network dynamics during the training and testing phases might disturb classification.

As readout units we used perceptrons with a threshold value of 0.0 for binary classification and linear perceptrons for continuous output approximation tasks. Initial connection strengths were uniformly chosen between –0.05 and 0.05. We trained these readout neurons at the end of each input presentation, primarily using the standard delta rule. In general just one neuron was used per readout, unless indicated otherwise. We used typical learning rates of 0.01 and 0.15 for respectively the delta and p-delta rules [2]. For all experiments we will report on when we used settings that deviate from these standards.

Chapter 7
Experiments and Results

Liquid State Machines seem to provide a powerful and noise robust method for temporal pattern recognition and input classification, as well as allowing feature extraction on continuous input streams. We will investigate the applicability and performance of this method by means of experiments specifically set up to test particular properties. The main goal of our experiments is to identify the general applicability of liquid state machines with spiking neural networks as a liquid for realistic noisy temporal pattern recognition. We will therefore address noise robustness and the required level of parameter tweaking for acquiring optimal performance of the setup in particular.

7.1 Temporal Activity Integration

With the implementation of our LSM completed, we set off with a first set of experiments to test whether it could indeed classify inputs. We chose to run a series of experiments with as simple as possible temporal input integration: classification of two independent Poisson random spike train generators with different spike generation rates. A Poisson spike train (PST) consists of a series of independent Poisson experiments to determine whether a spike occurs at that particular moment in time [25]. Due to this independent probabilistic nature single individual moments in time do not help in classifying the input stream.

The only variable in a Poisson experiment is the probability of a spike: on the long run the average number of spikes recorded will approximate this value. As there is no correlation between subsequent spikes, the Inter Spike Interval (ISI) property of the spike train also directly depends on the probability of a spike occurring. Due to the real-world nature of spike trains we do not refer to these spike probabilities directly but identify these by the average number of spikes over time in Hertz [38]. Together, this task requires the liquid to integrate the incoming spikes over an as long as possible period of time in order to allow accurate information on input averages to be extracted by the readout units.

The interesting property of this task is that we can calculate the maximum possible performance. Because only the number of spikes is informative on what generator was used, we know that LSMs with higher performance than chance are using this cue to make the classification. The best strategy would be to make the decision only dependent on whether the recent number of spikes is above

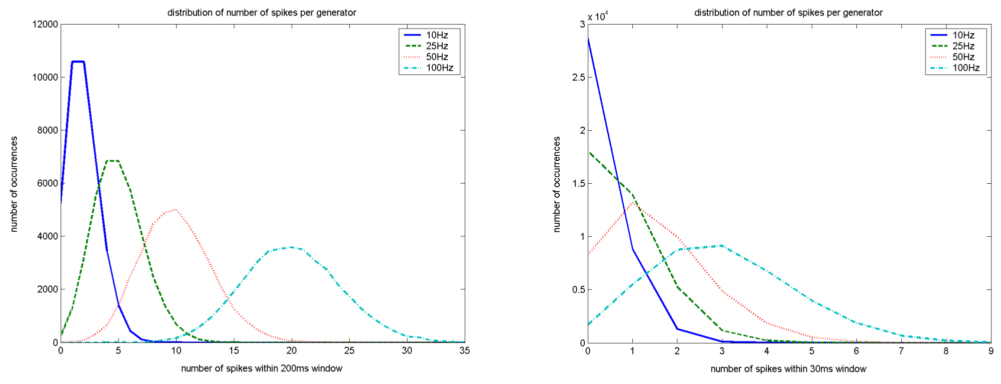

Figure 10. Average numbers of spikes generated by different random Poisson spike train generators within fixed intervals of 200ms (left) and 30ms (right). The only distinctive property for classification of these spike trains are their long-term averages, due to their stochastic nature no information is encoded in individual spike timings. Averages acquired over 50.000 runs per generator.

the cutting point of the two used distributions. If we take a look at the distribution of average number of spikes within a 200ms window in Figure 10 we see that the distribution 'bumps' of the 10Hz and 100Hz generators are nicely apart. We can calculate the maximum performance by dividing the sum of the correctly classified averages by the total number of occurrences. As there is very little overlap the maximum performance in this case would be 99%. However, if we check Figure 10 we see that the overlap within an averaging window of 30ms for these two generators is much greater. This means that if the liquid does not properly integrate much information in it's dynamics, the readout units are expected to only extract information from within the membrane time constant time of 30ms.

We tested three combinations of generators: 10Hz vs. 25Hz, 50Hz and 100Hz. Data was acquired from 50 independent runs per combination, 200ms per generator period, 1000 training and testing periods summing up to in total 400 seconds of simulated time per run. Though generating few spikes 10Hz and 25Hz are realistic spike generation frequencies. During training, at the end of the input stream we trained one perceptron to classify what generator was used. The liquid consisted of one column of 3x3x8 integrate-and-fire neurons gener-

10Hz vs.	**Th. max 200ms**	**Th. max 30ms**	**Max perf.**	**Avg. perf**	**Std.dev**
25Hz	79.9	63.3	73.0	60.0	6.1
50Hz	96.1	76.4	89.6	76.0	8.0
100Hz	99.9	88.9	96.6	87.5	5.9

Table 2. Theoretical maximum and actual attained performance in classification of two random Poisson spike train generators. Results achieved in 50 independent experiments per generator combination. Performance in percentage correctly classified patterns.

ated with synaptic upper bounds of $s_n=s_s=2.5 \cdot 10^{-7}$. These parameters have been experimentally found by exploration of a large part of the parameter space and were chosen as they achieved highest average and maximum performance, where other settings did not always surpass the theoretical 30ms maximum. Performance is measured in how often the readout unit correctly indicates the current generator class. We use such small values in order to work in realistic domains of micro Volt thresholds and nano Ampere inputs, but all could be scaled up to more accustomed values.

The results shown in Table 2 show us quite clearly that our LSM setup is indeed capable of correctly classifying input streams. We can see that the average performance is very close to the theoretical maximum with a 30ms integration period. If we take the standard deviation into account we can conclude that a good share of the generated networks do integrate information in the liquid dynamics, as their performance lies beyond what's possible to extract from just the low-pass filtered state of individual neurons. We double-checked this by running a separate set of experiments in which we set s_n to 0 to disable all internal network connections. The best individuals all approached the theoretical maximum performance for 30ms averaging windows, just as we would expect.

Another observation worth mentioning is that the maximum performance does not reach the theoretical maximum for the full 200ms input period. We ran series in which we varied these periods. Longer experiments (400ms periods, results not shown) show only slightly higher averages while maximum performance remains the same as with shorter integration periods. These longer time spans might help to cancel out residual noise in the network activity. Together this indicates that the network's fading memory-span lies below 200ms. The maximum performances relates to an integration span in the order of 60 ~ 90ms, 2 to 3 times the membrane time constant and integration period of the low-pass filter. We found no significant differences in performance between using a single perceptron with the delta rule and training 1, 3 or 5 parallel perceptrons using the parallel delta rule.

7.2 Temporal Input-timing Integration

With the knowledge that our LSM setup can indeed classify input streams and integrate information in its liquid dynamics, we were interested in its capabilities to use exact spike timings. To investigate this we chose to pit fixed Inter-Spike Interval (fISI) generators against the random Poisson generators we used in the previous experiment. Such a fixed inter-spike interval generators create randomly offset spike-trains of which the spikes follow each other with a fixed interval. The random offset (together with activity residue of previous generators) ensures that the readout units cannot just learn a fixed end-state, but have to identify a certain liquid dynamic for classification. By setting the long-term average spike generation rates equal for both types of generators we made sure that the LSM could not use the same strategy of linearly separating input or liquid activity.

Average input rate	Th. max over 100ms	Maximum performance	Average performance	Standard deviation
25Hz	68.1	82.6	62.0	8.6
50Hz	67.7	80.4	59.7	7.0
100Hz	67.4	95.6	67.4	11.0

Table 3. Theoretical maximum and actual attained performance in classification of a random Poisson spike train generator and a fixed Inter-Spike Interval generator at the same average spiking rates. Results gathered in 50 independent runs per combination. Performance in percentage correctly classified patterns.

We tested three combinations of Poisson and ISI generators at 25Hz, 50Hz and 100Hz. Data was acquired from 50 independent runs per combination, 100ms per generator period, 1000 training and testing periods. The learning rate to train the single readout perceptron was 0.01, while the liquid consisted of one column of 4x4x5 integrate and fire neurons generated with s_n=s_s=$2.5 \cdot 10^{-7}$.

We can see in Table 3 that quite many runs achieved much higher performance than would be possible when relying solely on the linear separation of average input activity. This means that actual spike timing has been taken into account. The maximum performance found for identifying 100Hz generators is very high, but does drop slightly with lower fire frequencies; as the standard deviation of the random Poisson spike trains decreases the LSM only have the input timings as a cue for correct classification.

Even though we find quite a high standard deviation of the average performance, we see that on average quite many individually generated networks find a better strategy than just to rely on the average number of spikes: this would result in a performance of 50%. This means that actual spike timing is being taken into account. If we take a look at the maximum performance we see that this indeed must be the case as it's far beyond what a simple uniform network activity strategy could achieve. Further evidence to support this can be

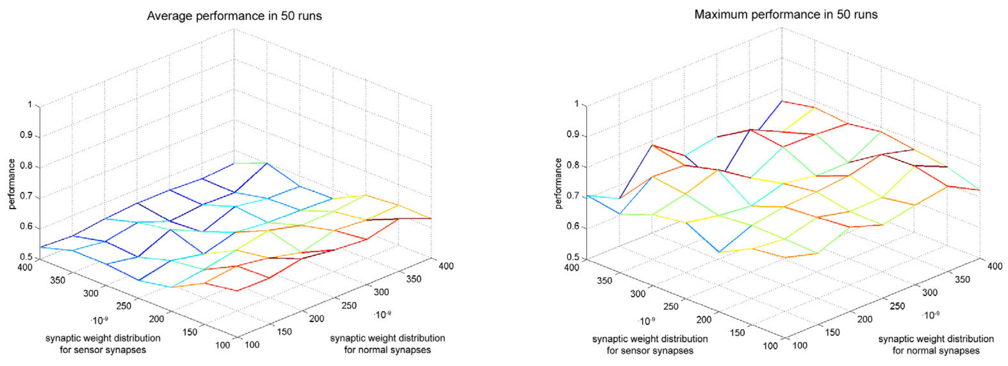

Figure 11. Average (left) and maximum (right) performance plots for classification of a 50Hz Poisson spike train and fixed inter-spike interval generators for different synaptic weight distributions. Synaptic weights are uniformly chosen between 0 and indicated values. While 'sensor' synapses receive external pattern input, 'normal' synapses connect neurons in the liquid.

seen in Figure 11; high values for the uniform generation of the sensory synaptic weights relate to a lower average performance. This is in accordance with the visual (see fig. 12) and statistical analysis of the liquid dynamics that such higher values seem to over-steer network activity, i.e. the network activity seems to follow the sensory input directly, instead of showing more homogenous activity patterns.

Also, the highest attained performances were all well beyond the theoretical maximum for 100ms averaging that worked in the previous experiment. This means that the integrate-and-fire networks did encode informative features on input timings in their dynamics. And of course, that the readout perceptrons were able to extract this information

7.3 Temporal Pattern Recognition

In the previous experiments we have shown that the LSM setup can be applied for the classification of randomly generated template based patterns. The 'templates' we used were very simple in nature: uniformly distributed and fixed interval spike trains. We will now up the ante by using more complex templates for spike train generation. Apart from testing the classification performance of such continuously generated patterns, we will also investigate with what performance fixed patterns can be detected in a stream. This latter task is expected to be much more simple as the readout unit can get away by recognizing just the (more-or-less) stable end state of the fixed pattern. In order to make the task more challenging, we chose to have the readout units differentiate the continuous stream from not just only one fixed pattern but also from three such fixed patterns.

As generator templates we used linearly increasing, decreasing and sinoid probability templates. The sinoid period was set to 60π, scaled to an amplitude of 0.05 and offset by 0.065. The linear generator was monotonously increasing from 0 to 0.2 over 250 time steps, after which it decreased to 0 again over an equal period. Input epochs were 250ms in length and each time the starting offset was chosen randomly. These function characteristics were chosen such that the resulting spike trains had a roughly equal average activity. It should be noted that these templates are very similar and that the class identification by the resulting spike trains is a very daunting task.

The fixed patterns were generated from the same continuously generated stream against which they should be classified. To avoid lucky shots in the sense of using simple identifiable sets of patterns

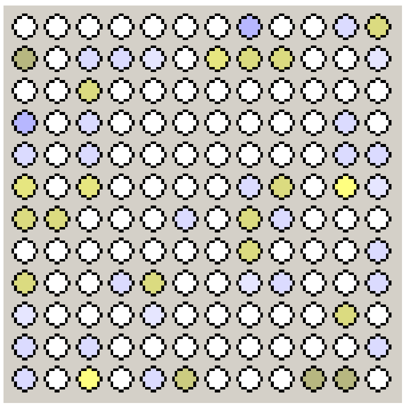

Figure 12. A typical liquid state snapshot of a network of 144 neurons. Average neuron activity is colour coded: the darker blue the more active the neuron was over 30ms. Yellow tinted neurons receive the input patterns over sensory synaptic connections. Internal connections are not shown.

Task Patterns	**Performance**			
	Maximum	**Average**	**Standard deviation**	**Minimal**
Random sinoid vs.				
1 fixed sinoid	100.0	99.4	1.3	91.2
3 fixed sinoids	96.4	81.3	11.0	52.8
3 fixed linear	99.7	94.5	5.0	72.0
1 random linear	78.0	67.3	7.6	50.6
Random linear vs.				
3 fixed sinoids	97.9	80.2	10.3	53.7
3 fixed linear	99.6	91.6	10.1	51.0

Table 4. Performance in binary classification of template-based generated input streams. Fixed patterns are re-used throughout the experiment, while random streams were continuously stochastically generated for each input presentation. Both streams had equal average activity and were randomly switched when after 200ms the classification decision had been made. The results of the fixed pattern experiments are the average of 10 independent experiments of 50 runs, each of those ten with a freshly generated set of fixed patterns.

by chance, we ran all these experiments 10 times in which a fresh set of fixed patterns was used. We pitted these fixed patterns against both the generator used to create them, as well as the 'other' generator (sinoid vs. linear). All experiments consisted of 50 individual runs that all used randomly generated networks for the liquid.

Performance was measured as the percentage of cases where the readout unit's output was 1 at the end of the input period of the continuous random generator and 0 otherwise. The results of these experiments can be read in Table 4. In contrast to the other performance tables we also provide the worst performing LSM, as this sheds more light on the complexity of the task and the ability of LSMs of identifying (a number of) fixed patterns in a continuous stream.

The results of these experiments show that, as expected, the liquid state machine framework is very suited for classification of individual patterns. The perfect maximum score and very high average performances found in the task of recognizing a single fixed pattern out of a continuous stream proofs the intuitive notion that it's relatively easy for the readout perceptrons to identify stable end-states.

The performances in the tasks of identifying a set of three different fixed patterns from a continuous generator are much more interesting. Table 4 shows the lowest maximum score, the overall average (500) and the lowest minimal performances found over ten separate experiments with different sets of fixed patterns. These results clearly indicate that the readout units are indeed capable of detecting stable features between a set of patterns, the maximum performances are near perfect for any combination of generator and set of fixed patterns.

When we compare these results with the experiment of classifying the random sinoid from the linear generator we see a drop in performance of around 25% correct. Though this is a significant fall, it was sure expected for this

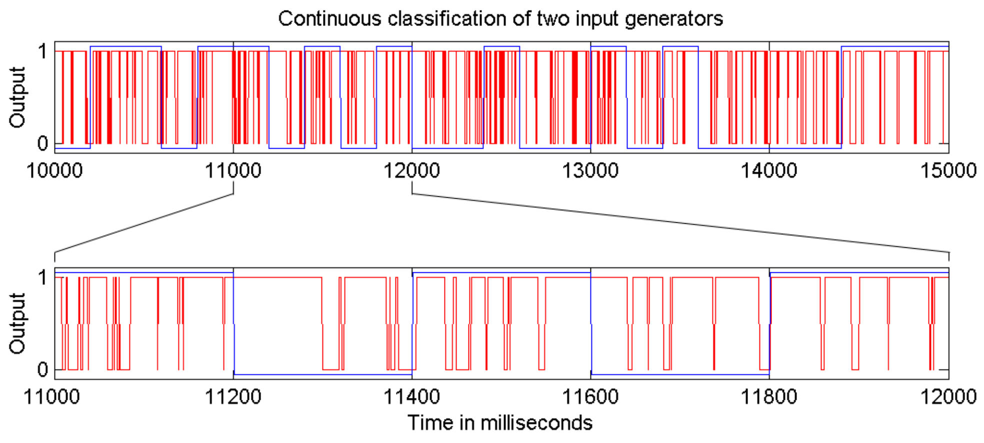

Figure 13. Continuous output plots of a readout unit Task was to recognize the randomly generated spike train, based on a sinoid probability function, by an output of zero and to identify one particular fixed pattern by an output of one. Readout consisted of a single perceptron trained using the p-delta rule and acquired a score of 99.6% correct over a 50.000ms testing period (2500 testing moments). Input class in blue (slightly offset and scaled for better visual comparison) and actual output in red. Bottom figure is a detail of top, zoomed in on time steps 11.000 to 12.000.

is a much more complicated task. It does show us, however, that template based spike pattern generators can be used for class identification and that we can expect high performance if we can make these to use specific spike timings as these lead to more identifiable end-states of the liquid.

7.4 Continuous Readouts

All experiments we have presented till now have only been aimed at investigating the possibilities of using the LSM framework with networks of integrate-and-fire neurons for identifying patterns after a fixed input period. The readout units were trained to recognize stable features in the state of the liquid at the end of the input, not to provide correct output during all of this period. Though handy, it would not be realistic to expect correct immediate classification; the identifying features simply lie in the temporal structure of the input. In fact, certain patterns might only be recognizable after a relatively long input period.

There are three important questions if we are interested in the continuous classification of input patterns. The necessary length of input before a reliable decision can be made, the stability and reliability of the output of the readout units over time and what the time-span is during which the readouts provide reliable and correct classification. This last question is particularly important as we train our readout units at fixed intervals. The first question is extremely task dependent and therefore not interesting for our investigation. The other two questions can be addressed though by taking a look at the continuous output of readout units.

Figure 13 shows the binary output of a perceptron trained to recognize a fixed spike train from a continuous random stream each 200ms, both streams based on the same sinoid template used in the previous experiment. Perform-

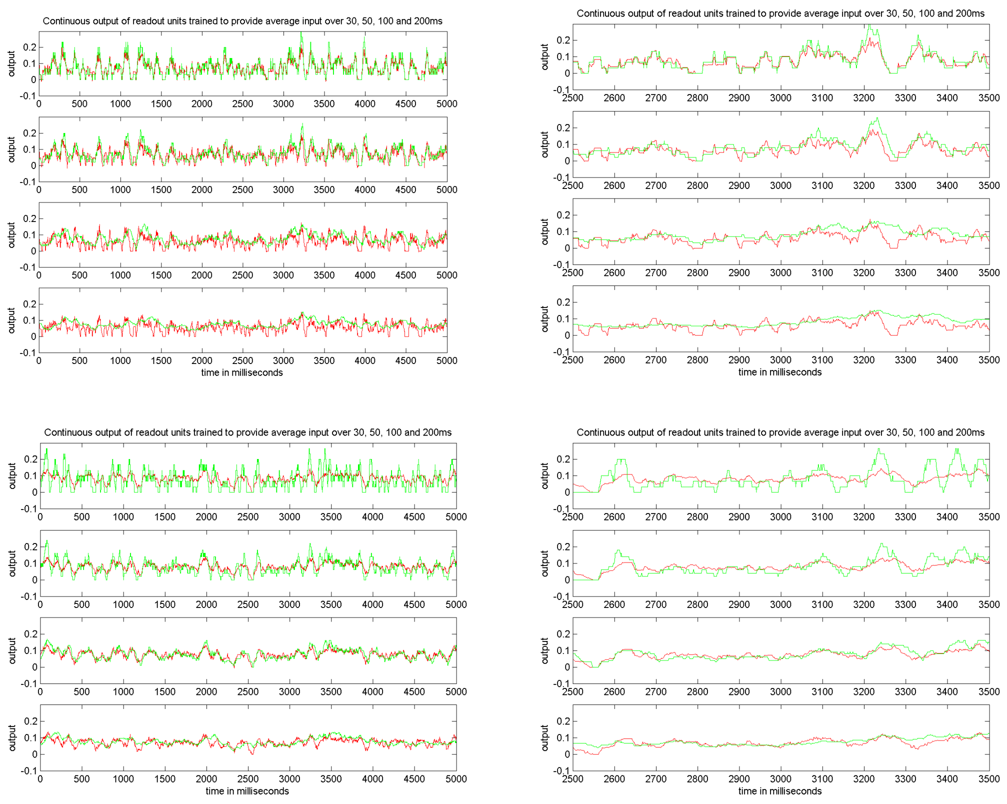

Figure 14. Continuous output plots for readout units trained to provide the average input over different integration periods. Respectively (top to bottom) 30ms, 50ms, 100ms and 200ms. Liquid activation was presented to the readout units low-pass filtered over 30ms (top row) and 80ms. Two rightmost pictures are details of the left counterparts. Output from readout units in red, target values in green.

ance of this particular single perceptron trained using the parallel delta rule was 99.6% over 2500 test cases. Run time of the experiment was much longer than plotted in Figure 13, but would not have been much more informative.

The output of the readout unit is very unstable over time. It neighs more to '1' while dipping to '0' for shorter periods of time at seemingly random intervals. However, output is regularly stable for about 30ms just before the readout moments of a class '0'. This may be correlated with the low-pass output filter, but not fully. Though they lengthened, these stable periods did not surpass much beyond 40 to 50ms. These periods of stable output indicates that the network activity around these moments has identifiable stable features that are picked up by the readout unit. The readout units cannot be used for continuously identifying what pattern is being presented, as their performance is only reliable after 200ms of input. Other examples (with lower performance) provided a more stable output of '0' with occasional short jumps to '1' at appropriate readout times.

We ran a vast set of experiments in which we attempted to enhance the duration of reliable pattern classification. We trained readout units not only at the end of each input period but at fixed intervals of 10, 25, 50 or 100 time steps. Results were simply bad for the training intervals lower than 100ms. This can be explained by the liquid dynamics being too dynamic for the readout units to pick up stable features during the full input presentation. Using multiple (1,3,5) parallel perceptrons only improved performance a couple percent points above chance, much less than would be expected of these theoretically very powerful readout mechanisms. Using a single perceptron for classification and training intervals of 100ms for input periods of 200ms showed an overall drop in performance to a maximum of 88.4% and an average of 78.6%, but did provide more stable and reliable continuous readout outputs. Generally the output switched to the right classification after 100ms of input and remained relatively stable for the duration of the input stream.

Besides classification of input, it is possible to do other forms of computation on continuous input streams [48,50]. As we are currently investigating pattern recognition on single input streams the number of useful operations is limited. However, without going too far out of our way we can gain extra insight in the temporal integration properties and therefore memory-span of the liquid by requiring a set of readout units to approximate the average input over different periods of time.

In this set of experiments we added four extra linear neurons that were trained to provide the input average over 30, 50, 100 and 200ms. We ran these experiments with both 30ms and 80ms as the integration periods for the low-pass filter. The four plots in Figure 13 show the response of these readout units after training, along with their target value plotted in green. The plots in the top row were taken from experiments that provided the standard 30ms low-pass filtered liquid state to the readout units, in the experiment shown in bottom row an averaging window of 80ms was used.

These plots give good insight in the relation between the integration period of the low-pass filter and the amount of dynamics the readout units get presented: the shorter the averaging window the more chaotic the input and because these are simple linear neurons also their output. The readout units that get the 30ms average liquid state can approximate the input average very well over 30ms and 50ms and increasingly less so for longer periods. Oppositely, the 80ms averaging filter allows the readouts to approximate the average input activity over 100ms very well and reasonably for 200ms, but removes much of the liquid dynamics that allow these linear neurons to pick up the short term input averages.

Another obvious conclusion we can draw from these plots is that all readout rely on the same features, as their output is roughly the same after training, only scaled differently. Inspection of their weight vectors shows that these are not uniformly distributed and that overall liquid activity is not used as a queue per se. This is in check with the sensory synaptic connections not controlling their post-synaptic neurons completely and the observation that network activity remains for a good while (>100ms) after input has seized.

The main conclusion we have to draw is that the networks we used as liquid integrate the average input very well, even over time-spans much longer than can be expected from the membrane time constants of the individual neurons. The reason for the readout units not to approximate these averages very well beyond the integration period of the input filter is the linear nature of these

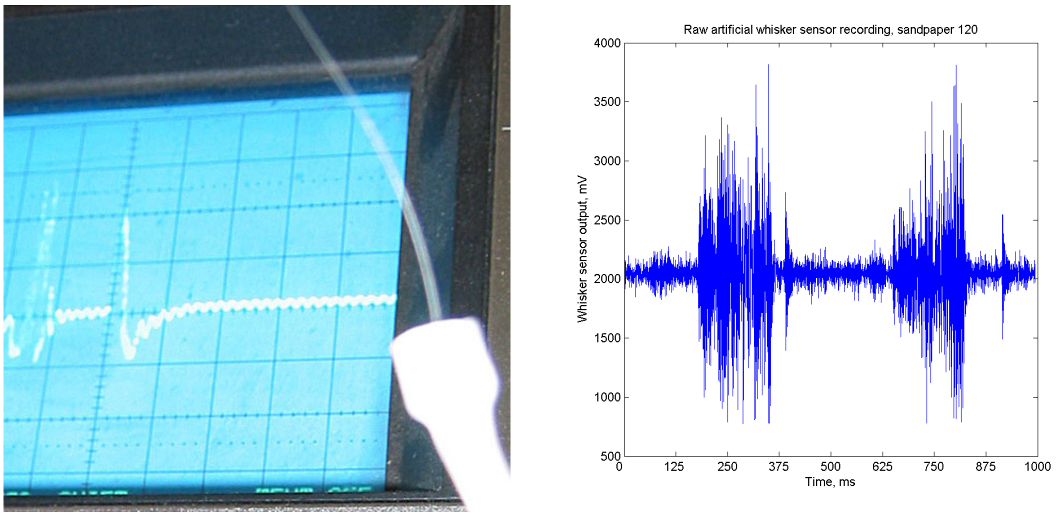

Figure 15. (Left) An artificial whisker sensor and its signal on the oscilloscope screen. The sensor consists of a strand (here a real rat whisker) glued to the membrane of a small electro microphone. The signal is amplified before being fed to the A/D converter that sends it to the computer that records the signal. (Right) Recording of a full sweep (back and forth) of an artificial whisker sensor against a piece of sandpaper (grain 120).

readouts combined with the input-filter itself. In case of the 80ms filter the short-term dynamics are effectively filtered out, while the 30ms filter does not provide enough stable features that the single perceptron can rely on. Using a low-pass filter to translate the liquid state is seemingly too coarse for fine-grained tasks.

Though we will cover the application of dynamic synapses in the liquid more thoroughly later on, we should remark here that in this set of experiments these synapses tended to make the networks a tad too dynamic for continuous feature extraction. We were unable to train perceptrons to approximate the average input. The output of trained readouts remained completely chaotic. Visual inspection of the liquid did reveal more 'waves' of activity in the liquid. As fixed-interval classification performance did not drop significantly, we recon that the low-pass output filter might not translate the liquid state properly for continuous readouts.

7.5 Artificial Whisker-data Pattern Recognition

All experiments presented above used only simple stochastically generated input streams. While these spike trains did help us to provide insight in the possibilities of using liquid state machines in temporal pattern recognition, they were not very realistic input. Apart from the Hopfield-Brody speech recognition task [31,32] we are not aware of any attempt of using LSMs to classify patterns of real world data. Though speech recognition might proof very useful in the future, we are not very much interested in dealing with this problem using our LSM setup. Speech can be analysed very well in chunks, from which its many high-level features can be extracted [15]. In our opinion the LSM has many strengths that enable it to be applied to raw input streams, which is why we propose to use this method for near real-time identification of textures using artificial whisker sensors.

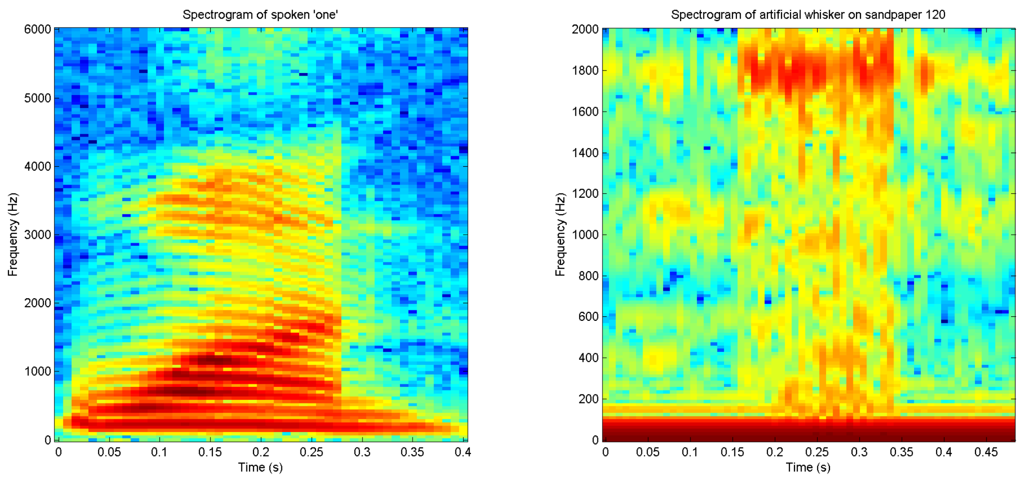

Figure 16. Spectral analysis of recording of the word 'one' (left) and the whisker sensor response to a single sweep over sandpaper (right, raw data shown in fig. 15). While human speech has many nice properties that help temporal processing, like different on and offsets on various frequency bands, audio processing of whisker recordings suffers from very high levels of noise and very few identifying features.

The ambitious European Union IST project 'Artificial Mouse' aims at the development of an 'artificial mouse': "an autonomous system that combines visual and tactile information processing to achieve object recognition, navigation and memory formation". The Zurich AI Lab is one of the labs that collaborate to build this robot and focuses primarily on artificial whiskers [8,44] and sensory integration. They built a series of artificial whisker sensors (see fig. 15), basically by attaching some random thread or a whisker from a cat or rat to the membrane of a small electro microphone. Statistical analysis of the signals acquired by brushing these sensors against various textures indicates (see fig. 15) these can indeed be discriminated [27]. Though a receptive field method has been developed, the project is currently looking into methods of higher biologically realism and in-tune with neurobiology data found in experiments with real rats: LSMs provide a suited means of potentially real-time texture recognition, definitely more biologically plausible than (off-line) statistical analysis. Also, part of the rat brain involved in texture recognition seems to resemble cortical micro-columns, groups of heterogeneous directed neurons that process information [13,56].

Though these recordings are very easy for humans to distinguish by just hearing them, they do proof to be hard to discriminate using known techniques. At the Zurich Institute for Neuro-Informatics attempts have been made to base classification on overall frequency power, but performance did not improve much beyond chance (Weiller, D., personal comm.). The signals are very noisy and unlike small bits of speech do not show nice blobs in their spectrograms (see fig. 16) but a seemingly rather randomly distributed series of peaks across many frequency bands. Also, as there are no clear onsets, peaks and offsets in these spectrograms this more or less disqualifies the method Hopfield [32] used to transform recordings into spike-trains.

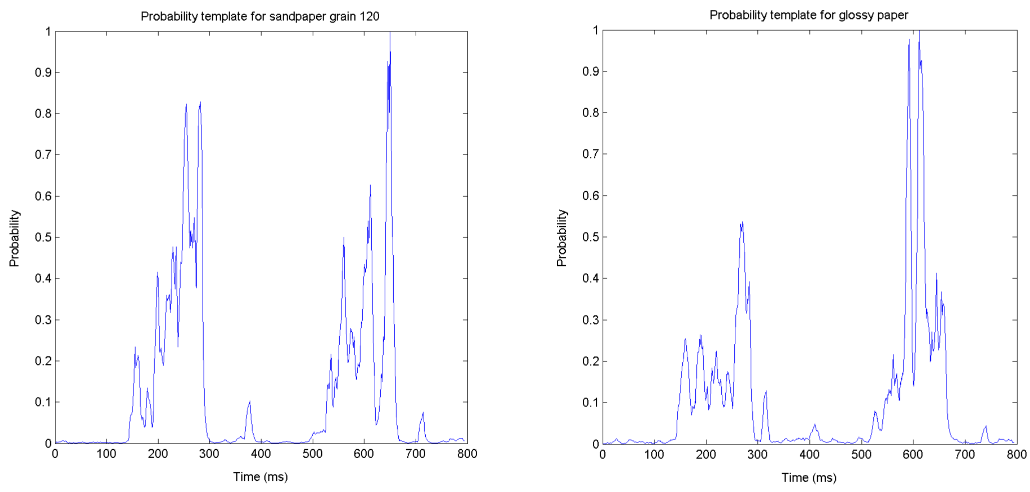

Figure 17. Temporal probability templates for spike train generation. Templates were generated from single frequency power bands of recordings of artificial whisker sensors swept back and forth (two peaks) against a certain texture. The raw data is easily identified by hearing it, but very ambiguous in all sorts of statistical analysis. Though very similar, the resulting spike trains could be correctly classified by our LSM setup.

Because no suited method for generating spike trains from noisy data was available and our stochastic spike-train templates worked very well in the previous tasks we decided to try and build such probability templates for a set of different textures [10]. Miriam Fend generously provided us with the data from long experiments where an artificial whisker was being swept back and forth by a small motor against a large set of textures. Each experiment was done with motor speeds of 1Hz and 4Hz. Each experiment consisted of 100 sweeps back and forth against one texture. The analog data from the sensor and the motor commands were converted and recorded at 4KHz. The full data set covers 25 different textures.

Figure 15 shows the sensor output data from one full sweep against sandpaper (grain 120): even when there is no contact with the texture the sensor readouts have high levels of noise, as can be seen in the 'silent' parts of the sweep and in the lower frequency bands of the power spectrogram of this recording. These high levels of noise can ruin the whole purpose of building a template that holds the right temporal information. To build fair probability templates we therefore calculated the 'average' sweep over the full texture recording. Though reduced, these averages still show high amounts of noise. However, we can be certain that we have captured the most important temporal traits of the full recording in this average sweep. The remaining levels of noise should not necessarily be problematic, as these are uniform and will be present in all recordings.

We analysed these average sweeps using power spectrographs (see fig. 16). While these seemed to be very ambiguous at first sight, we did notice a number of temporal differences in the more coarse spectrographs. When investigating the plots of single frequency bands we found peaks of different amplitude and timings, exactly what we need for generating a useable probability template.

Our first attempt for constructing these templates was to use the power spectrum of the frequency band between 1000 and 1200Hz. We did this by taking the power spectrograph with 10 bands of 200Hz each and allowing half-window overlap. From this full analysis we took the absolute value of the 6th band, as this type of analysis provides the data in imaginary numbers. To remove the highest levels of jitter we applied a smoothing filter with a window length of 12. Then we normalized between 0 and 1 after pronouncing the peaks more by taking the square. This method provided templates that we can use for the generation of stochastic spike trains and as can be seen in Figure 17 have identifiable temporal features as peaks.

We ran a series of experiments with spike trains generated using this probability template generation model. Instead of using the full template we decided to use only 200ms of the first sweep; we've shown in previous experiments that the memory span of our present liquid does not last over 200ms, the silent period between the back and forth sweep would kill all informative liquid activity. The templates were scaled such that the average number of spikes generated by these 200ms parts would be roughly equal.

Table 5 shows part of the results of this first set of experiments. Multiple texture templates were tested, but results were all very similar. We printed those of classifying glossy paper (GP) and sandpaper (grain 120, SP120) as these are very easy to differentiate by hearing. Their recordings, analysis and templates are very alike though. The performance of near 70% in classifying the continuously generated streams indicates that although our encoding method isn't perfect it does allow for good classification between the two patterns. The results in identifying fixed patterns are in line with our previous experiments with sinoid and linear templates.

Task Patterns	**Max. perf**	**Avg. perf.**	**Std. dev.**	**Min. perf.**
Fixed SP120 vs.				
1 fixed SP120	100.0	99.8	0.2	99.4
1 fixed GP	100.0	100.0	0.0	100.0
Random SP120 vs.				
1 fixed SP120	100.0	99.7	0.9	94.3
1 fixed GP	94.0	86.9	5.1	78.0
3 fixed SP120	93.0	82.0	8.7	56.5
1 random GP	69.3	57.8	5.0	51.0
1 random 60Hz	78.5	62.3	7.6	50.5

Table 5. Performance in binary classification of template-based generated input streams, in percentage correctly classified. Fixed patterns are re-used throughout the experiment, while random streams were continuously stochastically generated for each input presentation. Both streams had equal average activity and were randomly switched when after 200ms the classification decision had been made. The results of the fixed pattern experiments are the average of 10 independent experiments of 50 runs, each of those ten with a freshly generated set of fixed patterns.

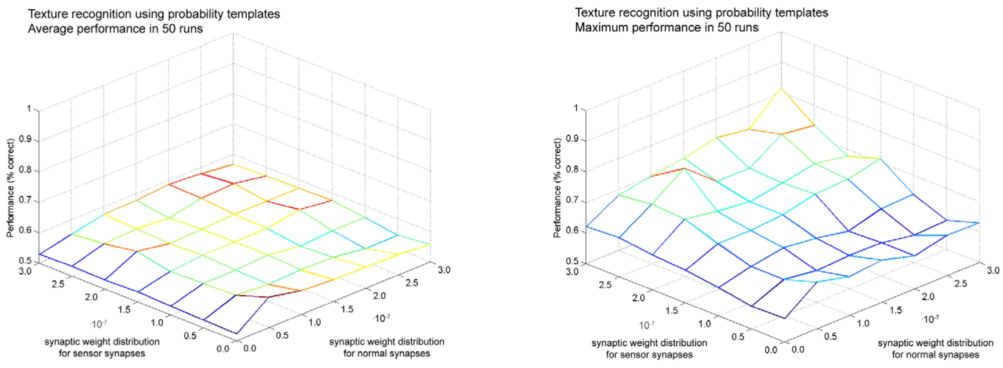

Figure 18. Average (left) and maximum (right) performance plots for classification of spike trains generated using probability templates of sandpaper (grain 240) and rough carton. Synaptic weights for connections to and within the liquid are uniformly chosen between 0 and indicated values. Sensor synapses receive external pattern input; normal synapses connect neurons in the liquid. Notice the difference in form of the both plots when compared to those of Fig 11. of the temporal integration task of the first experiment.

7.6 Liquid Optimisation

In order to try to increase the classification performance we investigated a large part of the parameter space of our LSM setup. We found striking differences in the effects of certain parameters when compared to the first two experiments. Figure 18 shows the effects of different combinations of synaptic efficacies for normal and sensory connections. It shows the importance of the strength of the sensory synapses to the maximum classification performance, a sharp contrast with our findings for the first two experiments (see fig 18.). Apparently the recognition of complex temporal and noisy patterns requires a neural pre-processor with different properties.

For all experiments we tested the effects of differently sized and shaped pre-processing networks. The LSM and ESM theory says that larger and more dynamic networks should lead to better performance. Intuition tells us its plausible that this generates more identifiable network states, allowing the readout units easier classification. Our findings contradict this notion. Figure 19 shows

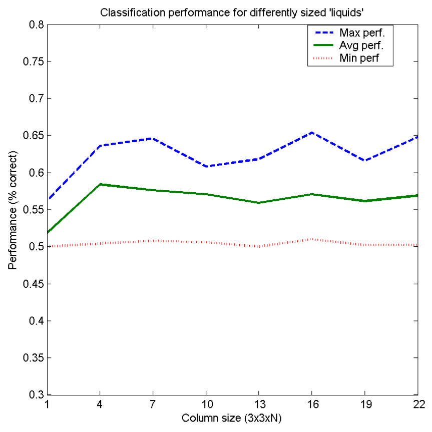

Figure 19. Performance in classification of whisker sensor texture recordings of fleece and cotton for differently sized pre-processing spiking neural networks. In all networks 20% of the neurons were provided with a sensory synapse. Small networks of just 9 neurons provide equally good pre-processing as large networks of up to 198 neurons, contradicting common beliefs about the computational powers of the 'liquid' networks.

rPST vs. fISI, 50Hz	**Normal**	**+STDP**	**+DS**	**+STDP +DS**
Maximum perf.	82.4	72.2	78.0	82.6
Average perf.	61.3	59.1	61.2	64.2
Std. Dev.	2.7	6.1	3.4	8.6
Minimal perf.	50.0	50.0	50.2	51.8

Table 6. Performance in classification of a random Poisson spike train (rPST) and randomly offset fixed inter-spike interval (fISI) generators, in percentage correctly classified. Normal indicates the usage of a 4x4x4 network of integrate-and-fire neurons using static synapses with weights uniformly distributed between 0 and $1.5 \cdot 10^{-7}$. Other columns provide results for when spike-timing dependent plasticity (+STDP) and/or dynamic synapses (+DS) were applied.

the acquired maximum, average and minimal performances over 50 runs in the classification of spike trains generated using probability templates created from the whisker recordings of fleece and cotton. It is striking that there is no correlation between the size of the networks and their resulting performance: even very small networks of 9 integrate-and-fire neurons function well enough as pre-processors for this complex task! For all our tasks we found that small networks could provide good classification if we provided 20% of the neurons with sensory synapses. Researchers at the Frauenhofer Institute found similar results when using an evolutionary approach to optimise the networks used in Echo State Machines. For a number of tasks performance increased with the network size, but for an equally large set of tasks (very) small networks provided equally good pre-processing (Van der Zant, personal comm.) as large networks.

We tested the effects of applying dynamic synapses [49] in the liquid networks for all experiments described above. We ran these experiments using both the parallel delta rule [2] and standard linear regression, and using the values for depression and facilitation Maass and Markram proposed [48]. Other than for the continuous readout task we found that the classification performance was virtually identical to that of networks with simple static weight synapses. Table 6 shows the performance on the inter-spike interval classification task of our second experiment. The average and minimal performance, as well as the standard deviation are virtually identical for both our normal static setup and that using dynamic synapses.

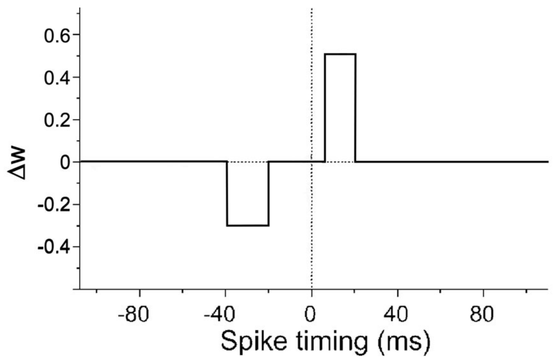

Figure 20. We can approximate the effect on the synaptic efficacy of differences in post and pre-synaptic firing times by simply using boxes. For network stability it is important that the LTD box is shallow and long, while the effect of LTP is stronger but only happens within a short window of time. Picture by Van Leeuwen, M. used with permission.

We also investigated the effects of dynamic synapses on the more complex texture recognition task. In the identification of sandpaper 120 and glossy paper the average maximum performance over 50 individual experiments was at 75.3% correct, opposed to the 69.3% correct for the setup with static synapses. Average and minimum performances remained the same though, as did the standard deviation. To achieve these results we used $s_s=1.0 \cdot 10^{-7}$ and $s_n=1.5 \cdot 10^{-7}$ and trained the single readout perceptron using the p-delta rule with a learning rate of 0.15.

To investigate our hypothesis that spike-timing dependent plasticity creates a network homeostasis that leads to better classification, we used a form of multiplicative box-form STDP [6]. In laymen terms this means that we prevent saturation of the weights by using a multiplicative form of weight change, avoiding the network fixation that many models of additive STDP allow. Box-form indicates that we approximate the real synaptic change (fig. 6) by simply choosing two boxes: one long and shallow one for LTD and one short and relatively high for LTP. Together this forms the following STDP equation to calculate the change in synaptic weight per combination of pre and post-synaptic spikes.

$$\Delta w = \begin{cases} A_{-} w W(\Delta t) & \Delta t < 0 \quad \text{LTD} \\ A_{+}(w_{\max} - w) W(\Delta t) & \Delta t > 0 \quad \text{LTP} \end{cases} \tag{6.1}$$

we use Δt to indicate the difference in timing of the spikes, A_{-} and A_{+} are the multiplicative constants for which we both use 0.01. Function W describes the form and height of the LTD and LTP (see fig. 20). We used a box lengths of 20ms (-40 to –20ms) and 15ms (5 to 20ms) with heights of respectively –0.3 and 0.5 for LTD and LTP.

The results of applying this type of STDP to the detection of the fixed inter-spike interval generator can be seen in Table 6. While on itself STDP does not have a particular good effect on performance, when combined with dynamic synapses it does lead to a slightly higher percentage of correctly classified generators. Especially when we take the standard deviation into account this change is significant. Table 7 shows the results of the texture recognition of a towel, classifying 7 fixed spike patterns from the continuous generator. Again,

1 random vs. 7 fixed Towel	**Normal**	**STDP**	**p-delta STDP**	**STDP DS**	**p-delta STDP DS**
Maximum perf.	86.8	78.2	71.7	86.6	87.8
Average perf.	68.3	67.1	67.1	71.6	79.1
Std. Dev.	5.7	3.4	1.5	12.6	10.7
Minimal perf.	54.8	53.8	63.7	50.3	50.4

Table 7. Performance in binary classification of template-based generated input streams, in percentage correctly classified. Both streams had equal average activity and were randomly switched when after 200ms the classification decision had been made. The results of the fixed pattern experiments are the average of 10 independent experiments of 50 runs, each of those ten with a freshly generated set of fixed patterns.

STDP alone has little good effect. Combined with the p-delta rule, however, it leads to very clustered performance over 50 independent runs. In combination with dynamic synapses the results are slightly better than for the static network. When the readout units are trained using the p-delta rule we find a massive increase in average performance of over 11%.

Only in these experiments we saw an increase in performance when using the p-delta rule for training the readout units. In all other experiments we did apply this rule, but never found radically better results. Though these results are preliminary, we can see the pattern that the combination of applying STDP, dynamic synapses and using the parallel-delta rule for training the readout units leads to significantly higher performance in the classification of complex patterns.

Chapter 8
Conclusions

We set up a full range of experiments to investigate the possibilities of using a Liquid State Machine with networks of randomly connected integrate-and-fire spiking neurons as 'liquid' pre-processor for temporal pattern recognition. Our first set of experiments further confirmed that these networks integrate information in their internal dynamics: the performance of the single-perceptron read-out units trained using the standard delta rule was beyond the theoretical maximum of what could be expected from just the averaging window length. These experiments showed that the fading short-term memory-span, or temporal integration of input activity period of these networks lays around 60 to 100ms. This shows that information is encoded in the network dynamics, as these time spans are well beyond the 30ms membrane time constants of the individual neurons. This conclusion is also backed by the observation that performance dropped significantly when we shut down the internal network dynamics. By using different types of input generators we confirmed that information about individual input timings is integrated. Together, this means that networks of spiking neurons can be used as pre-processors for temporal input and features can be extracted from their dynamic states by linear networks such as perceptrons.

8.1 Temporal Pattern Recognition

With the knowledge that temporal features and individual timings of input are integrated reliably we ran a series of experiments that aimed at investigating real temporal pattern classification. We chose to use probability templates to generate classes of spike trains. Instead of applying (temporal) jitter to fixed streams, templates capture the concept that class information is encoded in general temporal patterns. Some biological sensors are thought to work in similar probabilistic manner, but operate with population encoding to allow for more reliable input. We only investigated single input streams, though.

We used simple functions as probability templates and after training our setup could perfectly identify fixed patterns from other (sets of) fixed patterns. More importantly, with nearly equal performance it could differentiate fixed patterns from various continuous stream classes. Simple perceptrons prove to be good classifiers when coupled with good liquids: all best individuals could classify multiple fixed patterns against the continuous generator of the same class with performances of 96% correct classification and up. In this experiment however, the most interesting results were those of the classification of continuously generated input streams. Though significantly lower than for fixed patterns, scores of nearly 80% show that spiking neural networks can pre-process temporal input such that readout units as simple as single perceptrons are able to identify key dynamic features.

The weight vectors of the trained readout units show strong connections to a small number of neurons in the pre-processing network and mostly weak connections to the rest. This indicates that particular features have been identified for making the classification decision, instead of relying solely on general network activity.

8.2 Continuous Information Extraction

All tasks described above required input class identification by the readout units at fixed time intervals, at the end of the input stream. Exploration over time shows that their output is only reliable for roughly 30ms around these readout moments, but cannot be used for continuous classification. Attempts to train readouts for such a continuous task failed: the perceptrons are unable to follow full state trajectories.

However, the dynamic network state does allow for computation on the continuous input stream. We showed that linear neurons could indeed extract continuous features by training a group of readout neurons to provide the network's average input frequency over various time spans. The presentation of the liquid state to the readout neurons is very important. The averaging window length of the low-pass filter dictated the perceptron output dynamics, making approximation (far) beyond it impossible. Though visually identical, dynamic synapses made the network dynamics too chaotic to allow for proper approximation. Performance on continuous categorization however is on par with liquid networks using static synapses.

8.3 Texture Recognition

We took the frequency power over time of a broad (200Hz) band of recordings from artificial whisker sensors [19,44] being swept over various textures to create temporal probability templates. We used these templates to generate spike trains that correlate with the original textures. Our setup could perfectly identify fixed patterns generated from these templates from both opposing fixed and randomly generated streams. Again, our readout units could identify multiple patterns; this is not trivial, as these patterns were much more complex.

By using only 200ms of the full templates we acquired classification performances of continuously stochastically generated input streams of around 65 to 70% correctness. This performance is top notch in comparison with classification based on overall frequency power spectra.

8.4 Dynamic Liquid Optimisation

We applied a model of multiplicative box-form spike timing dependent-plasticity to enhance the separation property. This method altered the network weights based on the correlation between pre and post-synaptic activity. This leads to regulation of network homeostasis, as over-activity is depressed and correlated activity is stimulated. We found no improvements in the classification performance over networks with fixed synaptic weights. However, when applied to networks with very low synaptic weights, STDP could bring performance to optimal levels. It does add a set of extra parameters that need fine-tuning and adds significantly in the computational cost of network simulation.

We tested dynamic synapses [49] for all experiments described above and their effect failed to impress us. While the computational load of the simulations increased tremendously, classification performance of the readout units did generally not improve significantly. Dynamic synapses seem to make the liquid a tad too dynamic for continuous information extraction by the linear perceptrons. This might have everything to do with the low-pass filter we apply before we present the liquid state to the readout units; short-term dynamic trajectories might be dampened too much.

The preliminary results of the combination of STDP and dynamic synapses were very encouraging. We saw maximum performance levels increase slightly above those of static synapses, while the average performance improved between 5% and 10%. In the classification of the fixed inter-spike interval generator we noticed a significant increase in the minimum performance of 5%. All these results were attained using the p-delta rule for the training of the readout units. While in all other experiments did not lead to improved performance in these cases of more dynamic liquids the differences were striking.

8.5 LSM Parameter Tweaking

Throughout all the experiments we explored important parts of the parameter space of our LSM setup. From these results we distilled a set of settings that allowed us to run all of our tasks with satisfactory performance. It should be noted that these settings do not automatically lead to optimal classification performance.

Standard, we used static synapses with uniformly generated weights of upper bounds of $1.5 \cdot 10^{-7}$ for both sensory and normal connections. This is a very task-specific setting. The standard size of the networks we used was 150 neurons distributed on a 5x5x6 grid. Column shape did not proof to be of importance. With 20% of the neurons provided a sensory synaptic connection, even networks as small as 9 neurons did perform as well as any other liquid.

Network connectivity is very important however. While good results can still be achieved with lambda values of 2.0 we found that values of 1.2 lead to generally better performing set-ups. Lower connectivity leads to lower network activity and reduces chaotic effects in the network dynamics. Our analysis of this observation is that more local connectivity provides a more direct mapping from the input for the readout neurons: they are not required to 'see' through a lot of network dynamics.

Chapter 9 **Discussion**

Though strong theory has been developed about the computational power of spiking neurons, there have been few applications that really use their dynamics and pulse-coding capabilities. The liquid state machine is novel in the sense that it uses the non-linear dynamics of spiking neural networks instead of trying to tame them. As such, it is the first model in which these networks can be used for a broad range of computational tasks with continuous input, including temporal pattern recognition. We researched the usability of Liquid State Machines for temporal pattern recognition on continuous input streams. [45]. We trained readout units of single and parallel perceptrons using both linear regression and the parallel-delta rule to extract information from low-pass filtered snapshots of the state of the pre-processing 'liquid' network. We purposely did not use artificial problems as temporal XOR or the parity problem. While these are interesting from a computational perspective, we feel they would not provide us much insight in the applicability of our setup for real world temporal pattern recognition.

In our experiments we showed that information about both long-term temporal features and specific timings of the inputs are integrated in the internal dynamics of the liquid and the readout units can use these for classification. The recognition performance of fixed patterns was perfect. While visual inspection of the activity patterns in the liquid did not reveal any differences in the activity patterns, the readout units can be trained to extract features that identify the class of continuously random generated input streams with high performance. This allows for using the LSM framework in classification of patterns in real world data. We tested this for texture recognition with artificial whiskers [44]. Single broad frequency band power spectra can be used as stochastic temporal templates for generating spike trains. We achieved scores of up to 70% correct identification of these very alike and highly noisy inputs after pre-processing by randomly generated networks of spiking neurons.

However, it is necessary to point out that this randomness of these pre-processors is quite relative: the parameter ranges from which the values are drawn are slim. Generating liquids randomly seems to imply that the networks drawn from these distributions have similar computational powers, while they do not. While many of these parameters do have an effect on the average performance, the best individuals found all classify with roughly equal performance. Such 'good' networks apparently pre-process input much better for classification by the readout units, as their performance generally lies far beyond the average performance of the whole batch.

This indicates that the internal wiring of these networks is very important in creating the right type of network dynamics. We haven't found any obvious differences in the wiring or dynamics of 'good' and 'normal' networks. It should be investigated whether different settings or methods for connecting these networks could lead to more reliable generation of networks that do their pre-processing task well. Further, it would be interesting to see whether networks that pre-process classification information well are also optimised for other forms of computation on the input streams.

We explored large parts of the parameter space of our LSM setup and reported the settings that lead generally to good classification. Due to their input characteristics there seems to be no ultimate set of parameters that guarantees optimal performance. Apart from tweaking the settings for each task evolution could be employed for parameter optimisation. Classification performance could further possibly be optimised by using decaying or adaptive learn rates in the training of the readout units, instead of the fixed learn rates we used in our experiments. We incorporated these features in our experimentation software and we will report on our findings in the near future.

Our software, though not of the theoretically more efficient spike-event kind, allowed for running experiments very fast: in total we have simulated over a year of time just for the results presented. We have shown that networks of over 1000 spiking neurons can easily be simulated on standard desktop hardware. However, one of our more surprising findings was that very small networks of only 9 neurons often suffice for good pre-processing. This might only go for single input streams and that larger networks operate better at integrating information from more inputs and modalities. However, these findings are contradictory with the theory [34,35,48,49,51] that large networks and complex dynamics lead to better performance.

We showed that single perceptrons are unable to provide reliable continuous classification output based on low-pass filtered network states. Different methods of presentation of the liquid state to the readout units might allow for better classification and feature extraction; providing a set of average filters as input would allow the readout units to control their dynamics better. We think that continuous classification might be possible with just a single filter, by training a series of perceptrons at different(ly offset) intervals. A cascade of activity over these readout units would indicate the presence of a specific pattern. Parallel perceptrons could be suited for identifying state trajectories, though we did not find satisfactory results.

Further experiments will have to provide more insight in the applicability of LSMs for the classification of the high-resolution and complex temporal data generated by artificial whiskers. Our method of transforming the recordings into spike trains will have to be optimised to make better use of exact spike timings. Also, experiments with multiple input streams from a number of frequency bands will have to be run with more (identifying) information about the original texture it is expected that this will lead to higher performance in this complex pattern recognition task. Future work includes incorporating this method on a mobile robot [20,21] and using arrays of active whisker sensors [19] for real-time object and texture recognition.

We ran a set of preliminary experiments to explore the effects of optimising liquid dynamics using spike-timing dependent plasticity, dynamic synapses and the parallel delta-rule. Combined these three new techniques lead to considerably improved performance. When applied to a network of dynamic synapses STDP with synaptic redistribution could possibly even further increase the computational capabilities of the circuit [40]. We will investigate these cases in future work.

One particularly important question regarding LSMs is the necessity of using spiking neural networks for the liquid pre-processor. While being new, fresh and funky there is no apparent reason that recurrent neural networks with dynamic synapses could not provide equally useful dynamics at much lower computational costs [34]. Future experiments will have to compare these two different types of networks. Though the dynamics in the liquid are apparently very important [5], intuitively there have to be some stable invariant features in the liquid that the readout units can rely on, as there is no information to be retrieved from sheer chaos. Recent work shows that it is possible to extract invariants from a dynamic structure, a technique called slow-feature analysis [67]. Most work on such slowly varying feature extraction has been done on cells from the visual cortex, a brain region so highly specialized to its task that the technique might not be so easily applied to other dynamic structures [4]. It can best be described as an advanced, continuous form of principle component analysis (PCA) that can extract the behaviour of the more stable features using functions over time. This technique might be used to lower the computational cost during normal operation, as instead of a full spiking neural network the extracted stable features might provide enough information.

The results we've obtained show that the Liquid State Machine model is very promising for use in real world temporal pattern recognition. Fixed state and dynamic template based pattern recognition are just two of the strengths of this novel method. The main problems are the reliable generation of good preprocessing neural networks and the presentation of the liquid state to the readout units. A more general issue is the proper transformation of continuous analog input streams into spike trains. For use in AI and robotics it will be especially interesting to see how well LSMs can be used for the integration of different sensory modalities, one of the most difficult problems that exists in the field. Though its possibilities have not yet been fully explored, the Liquid State Machine provides a powerful new mechanism for temporal pattern recognition and real-time parallel computation on continuous input streams.

References

1. Abbot, L.F. & Nelson, S.B. *Synaptic plasticity: taming the beast,* Nature Neuroscience, vol. 3, pp.1178-1183 (2000).
2. Auer, P., Burgesteiner, H. M. & Maass, W. *The p-Delta Learning Rule for Parallel Perceptrons*, (2002).
3. Ayers, J., Davis, J.L., Rudolph, A. *Neurotechnology for Biomimetic Robots,* MIT press, Cambridge, MA. (2002).
4. Berkes, P. & Wiskott, L. *Slow feature analysis yields a rich repertoire of complex cell properties,* Cognitive Sciences, e-print archive vol. 2804 (2003).
5. Bertschinger, N. & Natschläger, T. *Real-Time Computation at the Edge of Chaos in Recurrent Neural Networks,* Journal of Neural Computation, vol. 16(7), pp.1413-1436 (2004).
6. Bi, G.Q. & Wang, H.X. *Temporal asymmetry in spike timing-dependent synaptic plasticity,* Physiology & Behavior, vol. 77, pp.551–555 (2002).
7. Bohte, S.M., Kok, J.N. & La Poutré, H. *Error-Backpropagation in Temporally Encoded Networks of Spiking Neurons*, Neurocomputing (2000).
8. Bovet, S., Fend, M. & Pfeifer, R. *Simulating Whisker Sensors – on the Role of Material Properties for Morphology, Behaviour and Evolution,* Proc. of the 8th Int. Conf. On the Simulation of Adaptive Behaviour (SAB), Los Angeles (2004).
9. Brecht, M., Preilowski, B. & Merzenich, M.M. *Functional architecture of the mystacial vibrissae*, Behavioral Brain Research, vol. 84, pp.81-97 (1997).
10. Buonomano, D.V. & Merzenich, M.M. *Temporal information transformed into a spatial code by a neural network with realistic properties*, Science, vol. 267, pp.1028-1030 (1998).
11. Buonomano, D.V. *Decoding Temporal Information: A Model Based On Short Term Synaptic Plasticity*, J. of Neuroscience, vol. 20(3), pp.1129-1141 (2000).
12. Buonomano, D.V. & Karmarkar, U.R. *How do we tell time?*, Neuroscientist, vol. 8(1), pp.42-51 (2002).
13. Carvell, G.E. & Simons, D.J. *Biometric analyses of vibrissal tactile discrimination in the rat*, J. of Neuroscience, vol. 10, pp.2638-2648 (1990).
14. Coren, S., Ward, L. & Enns, J. *Sensation and Perception,* 5th ed., Orlando, Harcourt (1999).
15. Deng, L. et al *Speech recognition using hidden Markov models with polynomial regression functions as non-stationary states,* IEEE Transactions on Speech and Audio Processing, vol. 2(4), pp.507-520 (1994).
16. Durbin, R., Eddy, S., Krogh, A., Mitchinson, G. *Biological sequence analysis,* Cambridge University Press, England (2002).
17. Edelman, G. *Neural Darwinism: The Theory of Neuronal Group Selection*, Basic Books, New York (1987).
18. Elman, J.L. *Finding Structure in Time,* Cognitive Science, vol. 14, pp.179-211 (1990).
19. Fend, M., Bovet, S., Yokoi, H. & Pfeifer, R. *An active artificial whisker array for texture discrimination*, Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Las Vegas, (2003).
20. Fend, M., Yokoi, H. & Pfeifer, R. *Optimal Morphology of a Biologically-Inspired Whisker Array on a Obstacle-Avoiding Robot,* Proc. of the 7th European Conf. On Artificial Life (ECAL), Dortmund (2003).
21. Fend, M., Bovet, S. & Hafner, V.V. *The Artificial Mouse – A Robot with Whiskers and Vision,* Proc. of the 35th Int. Symp. On Robotics (ISR), Paris (2004).
22. Fernando, C. & Sojakka, S. *Pattern recognition in a bucket: a real liquid brain*, ECAL (2003).

23. Floreano, D. & Mattiussi, C. *Evolution of Spiking Neural Controllers for Autonomous Vision-Based Robots,* in Gomi, T. (Ed.), *Evolutionary Robotics. From Intelligent Robotics to Artificial Life,* Springer Verlag, Tokyo (2001).
24. Franceschini, N., Pichon, J.M. & Blanes, C. *From insect vision to robot vision*, Phil. Trans. of the Royal Soc., London B, vol. 337, pp.283-294 (1992).
25. Gerstner, W. & Kistler, W. *Spiking Neuron Models,* Cambridge University Press (2002).
26. Goldenholz, D. *Liquid Computing: A Real Effect*, Technical report, Boston University Department of Biomedical Engineering, (2002).
27. Hafner, V.V., Fend, M., Lungarella, M., Pfeifer, R., König, P. & Körding, K. P. *Optimal coding for naturally occurring whisker deflections*, Proc. of the 10th International Conference on Neural Information Processing (ICONIP), (2003).
28. Häusler, S., Markram, H. & Maass, W. *Perspectives of the High Dimensional Dynamics of Neural Microcircuits from the Point of View of Low Dimensional Readouts*, (2003).
29. Hebb, D.O. *The organization of behavior,* Wiley, New York (1949).
30. Hochreiter, S. & Schmidhuber, J. *Long Short-Term Memory*, Neural Computation vol. 9(8), p.1735-1780 (1997).
31. Hopfield, J.J. & Brody, C.D. *What is a moment? "Cortical" sensory integration over a brief interval,* Proc. National Academy of Science USA, vol. 97. pp.13919-13924 (2000)
32. Hopfield, J.J. & Brody, C.D. *What is a moment? Transient synchrony as a collective mechanism for spatio-temporal integration,* Proc. National Academy of Science, vol. 98(3), pp.1282-1288 (2001).
33. Horne, B.G. & Giles, C.L. 'An experimental Comparison of Recurrent Neural Networks', In: Tesauro, G., Touretzky, D. & Leen, T. (eds), *Neural Information Processing Systems 7,* MIT Press, pp.697 (1995).
34. Jaeger, H. *The "echo state" approach to analysing and training recurrent neural networks*, GMD Report 148, German National Research Center for Information Technology, (2001).
35. Jaeger, H. *Short Term Memory in Echo State Networks*, GMD Report 152, German National Research Center for Information Technology, (2001).
36. Joshi, P. *Synthesis of a Liquid State Machine with Hopfield/Brody Transient Synchrony,* MSc. Thesis, Center for Advanced Computer Studies, University of Louisiana, Lafayette (2002).
37. Joshi, P. & Maass, W. *Movement Generation and Control with Generic Neural Microcircuits,* In: Ijspeert, A.J. et al (eds.), Proc. of the First International Workshop on Biologically Inspired Approaches to Advanced Information Technology (BIO-ADIT), pp.16-31 (2004).
38. Kandel, E.R., Schwartz, J.H. & Jessell, T.M. (Eds.) *Principles of Neural Science 3rd edition,* Appleton & Lange, Connecticut (1991).
39. Kempter, R., Gerstner, W. & Van Hemmen, J.L. *Hebbian learning and spiking neurons,* Physical Review E, vol. 59(4), pp.4498-4514 (1999).
40. Kempter, R., Gerstner, W. & Van Hemmen, J.L. *Intrinsic Stabilization of Output Rates by Spike-Based Hebbian Learning,* Neural Computation 13, pp.2709-2741 (2001).
41. Koopman, A.C.M., Van Leeuwen, M. & Vreeken, J. *Dynamic neural networks, comparing spiking circuits and LSTM*, Technical Report UU-CS-2003-007, Institute for Information and Computing Sciences, Utrecht University, (2003).
42. Koopman, A.C.M., Van Leeuwen, M. & Vreeken, J. *Exploring Temporal Memory of LSTM and Spiking Circuits,* IPCAT: Future of Neural Networks workshop, (2003).
43. Van Leeuwen, M., Koopman, A.C.M. & Vreeken, J. *Evolving vision-based navigation on wheeled robots,* Institute for Information and Computing Sciences, Utrecht University (2003).
44. Lungarella, M., Hafner, V.V., Pfeifer, R. & Yokoi, H. *Whisking: An Unexplored Sensory Modality*, From Animals to Animats 7 - Simulation of Adaptive Behaviour, (2002).

45. Maass, W. *Networks of Spiking Neurons: The Third Generation of Neural Network Models,* Neural Networks, vol. 10, pp.1659-1671 (1997).
46. Maass, W. & Bishop, C.M. (Eds.) *Pulsed Neural Networks*, MIT-press, Cambridge, MA, (1999).
47. Maass, W. & Zador, A.M. *Dynamic Stochastic Synapses as Computational Units,* Neural Computation, vol. 11, pp.903-917 (1999).
48. Maass, W., Natschläger, T. & Markram, H. *Real-time computing without stable states: A new framework for neural computation based on perturbations,* Neural Computation, vol. 14(11), pp.2531-2560 (2002).
49. Maass, W. & Markram, H. *Synapses as dynamic memory buffers*, Neural Networks, vol. 15, pp.55-161 (2002).
50. Maass, W., Natschläger, T. & Markram, H. *A Model for Real-Time Computation in Generic Neural Microcircuits,* in: Becker, S. et al (eds.), Proc. of NIPS 2002, Advances in Neural Information Processing Systems, vol. 15, pp.229-236. MIT Press, (2003).
51. Maass, W., Natschläger, T. & Markram, H. *Computational models for generic cortical microcircuits*, In J. Feng (ed), *Computational Neuroscience: A Comprehensive Approach*, chapter 18. CRC-Press (2004).
52. Natschäger, T., Markram, H. & Maass, W. *Computer models and analysis tools for neural microcirctuits,* in:Kötter, R. (ed.) *A practical guide to neuroscience databases and associated tools,* chapter 9, Kluwer Academic Publishing, Boston (2002).
53. Navy, G. et al *Regulation of releasable vesicle pool sizes by protein kinase A-dependent phosphorylation of SNAP-25,* Neuron, vol.41(3), pp.417-29 (2004).
54. Mehta, S.B. & Kleinfeld, D. *Frisking the Whiskers: Patterned Sensory Input in the Rat Vibrissa System,* Neuron, vol. 41, pp.181-184 (2004).
55. Pfeifer, R. & Scheier, C. *Understanding Intelligence*, MIT Press, Cambridge, MA (1999).
56. Prigg, T., Goldreich, D., Carvell, G.E. & Simons, D.J. *Texture discrimination and unit recordings in the rat whisker/barrel system*, Physiology & Behavior, vol. 77(4-5), pp.671-675 (2002).
57. Rabiner, L.R. *A tutorial on hidden Markov models and selected applications in speech recognition,* Proc. of the IEEE, vol. 77(2), pp.257-286 (1989).
58. Richardson, D. *A Discussion of simulation and training methods for a simple liquid state machine,* Technical report, Department of Biomedical Engineering, Boston University (2002).
59. Rumelhart, D.E., Hinton, G.E. & Williams, R.J. *Learning representations by back-propagating errors,* Nature, vol. 323 (1986).
60. Schmidhuber, J. & Hochreiter, S. *Guessing can Outperform many Long Time Lag Algorithms,* Technical note IDSIA-19-96 (1996).
61. Song, S., Miller, K.D. & Abbot, L.F. *Competitive Hebbian learning through spike-timing-dependent synaptic plasticity,* Nature Neuroscience, vol. 3, pp.919-926 (2000).
62. Thorpe, S., Delorme, A. & Van Rullen, R. *Spike based strategies for rapid processing*, Neural Networks, vol. 14(6-7), pp.715-726 (2001).
63. Vreeken, J. *Spiking neural networks, an introduction*, Technical report UU-CS-2003-008, Institute for Information and Computing Sciences, Utrecht University (2003).
64. Wang, D. & Terman, D. *Image Segmentation Based on Oscillatory Correlation,* Neural Computation, vol. 9, pp.805-836 (1997).
65. Williams, R.J. & Zipser, D. *A Learning Algorithm for Continually Running Recurrent Neural Networks*, Neural Computation, vol. 1, pp.270-280 (1989).
66. Williams, R.J. & Peng, J. *An Efficient Gradient-Based Algorithm for online Training of Recurrent Neural Network Trajectories*, Neural Computation, vol. 2, pp.490-501 (1990).
67. Wiskott, L. & Sejnowski, T.J. *Slow Feature Analysis: Unsupervised Learning of Invariances,* Neural Computation, vol. 14(4), pp.715-770 (2002).
68. Synapse illustration: http://www.ship.edu/~cgboeree/synapse.gif