Available online at www.sciencedirect.com
ScienceDirect
Neural Networks 20 (2007) 391–403
Neural Networks
www.elsevier.com/locate/neunet

2007 Special Issue

An experimental unification of reservoir computing methods

D. Verstraeten*, B. Schrauwen, M. D'Haene, D. Stroobandt

Department of Electronics and Information Systems, Ghent University, Sint-Pietersnieuwstraat 41, 9000 Gent, Belgium

Abstract

Three different uses of a recurrent neural network (RNN) as a reservoir that is not trained but instead read out by a simple external classification layer have been described in the literature: Liquid State Machines (LSMs), Echo State Networks (ESNs) and the Backpropagation Decorrelation (BPDC) learning rule. Individual descriptions of these techniques exist, but a overview is still lacking. Here, we present a series of experimental results that compares all three implementations, and draw conclusions about the relation between a broad range of reservoir parameters and network dynamics, memory, node complexity and performance on a variety of benchmark tests with different characteristics. Next, we introduce a new measure for the reservoir dynamics based on Lyapunov exponents. Unlike previous measures in the literature, this measure is dependent on the dynamics of the reservoir in response to the inputs, and in the cases we tried, it indicates an optimal value for the global scaling of the weight matrix, irrespective of the standard measures. We also describe the Reservoir Computing Toolbox that was used for these experiments, which implements all the types of Reservoir Computing and allows the easy simulation of a wide range of reservoir topologies for a number of benchmarks.
© 2007 Elsevier Ltd. All rights reserved.

Keywords: Reservoir computing; Memory capability; Chaos; Lyapunov exponent

1. Introduction

Recurrent neural networks (RNNs) seem to offer an attractive method for solving complicated engineering tasks. They have the advantages of feedforward networks, which include robustness, learning by example and the ability to model highly nonlinear systems, and add to that an inherent temporal processing capability. Possible – and actual – applications are manifold and include the learning of context free and context sensitive languages (Gers & Schmidhuber, 2001; Rodriguez, 2001), control and modeling of complex dynamical systems (Suykens, Vandewalle, & De Moor, 1996) and speech recognition (Graves, Eck, Beringer, & Schmidhuber, 2004; Robinson, 1994).

RNNs have been shown to be Turing equivalent (Kilian & Siegelmann, 1996) for common activation functions and can approximate arbitrary finite state automata (Omlin & Giles, 1994). So, theoretically, RNNs are very powerful tools for solving complex temporal machine learning tasks. Nonetheless, several factors still hinder the large scale deployment of RNNs in practical applications. So far, not many learning rules exist (Haykin, 1999; Jaeger, 2002; Suykens & Vandewalle, 1998) and most suffer from slow convergence rates, thus limiting their applicability. Recently, however, three similar solutions for this problem have been proposed independently.

Maass et al. (Maass, Natschläger, & Markram, 2002), Jaeger (2001a) and Steil (2004) all describe the possibility of using an RNN without adapting the weights of the internal connections. In principle, the output can be generated using any type of classifier or regressor, ranging from a perceptron (Minsky & Papert, 1969) to a Support Vector Machine (Vapnik, 1995). In almost all applications, however, a simple linear discriminant or regression algorithm (Duda, Hart, & Stork, 2001) is used to compute the desired output. This type of readout function offers some quite convincing advantages – such as its ease of training and guaranteed optimality in a least squares sense – and yields very good results. From this viewpoint, the function of the reservoir is similar to that of a kernel in the case of kernel-based methods, by projecting the inputs into a high-dimensional space which enhances the separability. For traditional methods, the projection into the high-dimensional space does not need to be computed because of a convenient

* Corresponding author. Tel.: +32 9 264 34 04; fax: +32 9 264 35 94.
E-mail address: david.verstraeten@ugent.be (D. Verstraeten).
URL: http://www.elis.ugent.be/SNN (D. Verstraeten).

0893-6080/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2007.04.003

mathematical construction (the so-called *kernel trick*), while for reservoir methods the projection is explicit. A significant advantage of the reservoir approach, however, is the fact that the kernel is able to incorporate temporal information present in the inputs.

1.1. Using RNNs as reservoirs

The *Liquid State Machine* (LSM) by Maass et al. (2002) proposes a generic framework, where the reservoir can consist of a broad range of node types, and needs to obey a quite unrestrictive property (the *point-wise separation* property) (Maass et al., 2002) in order to be computationally useful. In practice, most descriptions of the LSM use reservoirs built from a relatively simple spiking neuron model called the Leaky Integrate and Fire (LIF) neuron (Maass & Bishop, 2001) with a dynamic synapse model (Gerstner & Kistler, 2002), but the use of threshold logic gates (TLGs) has also been described (Natschläger, Bertschinger, & Legenstein, 2004). Spiking neuron models are more complex than sigmoidal neurons but they have been shown to be computationally more powerful (Maass, 1997). The readout layer, on the other hand, is also very broadly specified: it needs to be able to approximate any continuous function over a compact set $X \subseteq \mathbb{R}^N$ of input values. When both conditions are fulfilled, the resulting LSM is guaranteed to be able to approximate any time-invariant function with fading memory[1] operating on timeseries. However, these sufficient conditions are too broad to offer a practical guideline for constructing reservoirs, and quite probably universal approximation of temporal functions is not needed to solve real world problems. So, while this result is very convincing from a theoretical point of view, the transition to practical applications is still unclear.

Jaeger independently proposed a very similar computational idea which he calls *Echo State Networks* (Jaeger, 2001a). The main difference with LSMs lies in the type of nodes that constitute the reservoir: where LSMs are usually built from spiking LIF neurons, these reservoirs are built from analog sigmoidal neurons. Here too, a theoretical property is defined for potentially interesting reservoirs called the *echo state* property (Jaeger, 2001a), which expresses – informally stated – the fact that the influence of inputs on reservoir states fades away gradually. Further, an upper and lower bound are defined for the echo state property that are very easy to compute and depend only on the weight matrix of the reservoirs. However, this property alone is not sufficient to guarantee optimal performance for a given problem, and the search for a good reservoir requires experience and can take some time. In Jaeger (2002), some guidelines are offered, but a structured method is still lacking.

Thirdly, Steil proposes an O(N) learning rule for RNNs called *Backpropagation Decorrelation* (BPDC) (Steil, 2004). The BPDC rule is an extension of a state-of-the-art learning algorithm for RNNs (Atiya–Parlos recurrent learning (Atiya & Parlos, 2000)). It appears that this APRL learning rule restricts the adaptation of the weights to the output layer, effectively splitting the RNN into a reservoir and a readout layer. Only the weights to the output layer and the recurrent connections inside the output layer are trained. Thus, the use of an RNN as a reservoir was attained here from an entirely different angle than the previous two approaches. Here too, sigmoidal neurons are used, but a significant difference between BPDC reservoirs and ESNs is the fact that feedback connections from the readout layer into the reservoir and into the readout layer itself are used, whereas in practice this is hardly ever the case for ESNs.

1.2. Applications of reservoir computing

Several successful applications of reservoir computing to both ‘abstract’ and real world engineering applications have been reported in the literature. Abstract applications include dynamic pattern classification (Jaeger, 2001b), autonomous sine generation (Jaeger, 2001a; Verstraeten, Schrauwen, & Stroobandt, 2005) or the computation of highly nonlinear functions on the instantaneous rates of spike trains (Maass, Natschlger, & Markram, 2004). In robotics, LSMs have been used to control a simulated robot arm (Joshi & Maass, 2004), to model an existing robot controller (Burgsteiner, 2005b), to perform object tracking and motion prediction (Burgsteiner, 2005a; Maass, Legenstein, & Markram, in press) or event detection (Jaeger, 2005). ESNs have been used in the context of reinforcement learning (Bush & Anderson, 2005). Also, applications in the field of Digital Signal Processing (DSP) have been quite successful, such as speech recognition (Maass, Natschläger, & Markram, 2003; Skowronski & Harris, 2006; Verstraeten, Schrauwen, Stroobandt, & Van Campenhout, 2005) or noise modeling (Jaeger & Haas, 2004). Finally, the use of reservoirs for chaotic timeseries generation and prediction has been reported in Jaeger (2001b, 2003), Steil (2005, 2006).

1.3. Comparing reservoir computing methods

In this contribution, we offer an experimental overview of the different implementations of reservoir computing described in the literature. In Section 2, we present experimental results for benchmarks with different characteristics. Section 2.1 shows the relation between the complexity of the reservoir nodes and the performance on three tasks. Section 2.2 investigates the effects of memory on the node level and the reservoir level. Thirdly, Section 2.3 links three existing bounds for the echo state property to the actual dynamics in the network, quantified using a Lyapunov-inspired method as a measure of the network dynamics. In Section 3, we present a novel toolbox we developed to do this work, that incorporates all reservoir implementations currently described and some novel ones. We outline its modular structure and describe which components are present. In Section 4, finally, we conclude and outline future work.

[1] This means that the current output depends on inputs from a finite time window in the past.

2. Experimental unification of reservoir-based methods

Contributions on reservoir computing methods often focus on a specific type of reservoir interconnection and node type. LSMs focus on a specific, small world interconnectivity pattern with reservoirs built from spiking neurons. ESNs, on the other hand, have been described with several different interconnection structures and analog neurons. What is still lacking from the literature, however, is a thorough overview of the performance of different reservoir node types.

When one wants to solve a task using reservoir computing, one has to choose not only the node type but also the interconnection topology and weight distribution. Several metrics have been described in the literature that are supposed to offer an a priori indication of how the reservoir will perform without explicitly having to apply it to a task. However, a clear indication of which metric values are optimal for which task is not yet available, and the values of these metrics have not yet been linked to the actual reservoir dynamics.

In this section, we present experimental work that gives a broad overview of the influence of certain reservoir parameters on the performance and the network dynamics. To account for the variability of the performance due to the stochastic construction of the reservoirs, we simulated every parameter setting 30 times. We used three tasks with very different characteristics to evaluate the performance of reservoirs, which we will briefly describe here. See Appendix A for a thorough description of the experiments described in this contribution.

The first task is a rather complex classification problem, namely the isolated spoken digit recognition task discussed in Verstraeten et al. (2005). The task is to recognize ten isolated digits spoken by five different female speakers. Performance is expressed as the word error rate (WER) and is measured using ten-fold cross-validation over the dataset containing 500 samples. The speech was preprocessed using a biological model of the human cochlea by Lyon (1982) resulting in a 77-channel cochleagram. For spiking reservoirs, this cochleagram was converted into spike trains using a filter-coding method called BSA (Schrauwen & Van Campenhout, 2003).

The second task is an evaluation of the memory capacity of the reservoir, where the task is to reproduce delayed versions of a random uniform noise signal. This task is described in Jaeger (2001b). The delays range from 1 to 100 timesteps. The memory capacity is measured as the total amount of variability in the output of the classifiers that was caused by the input to the reservoir (see Appendix A and Jaeger (2001b) for a precise definition), and is therefore an indication of the amount of information contained in the input that is still present inside the reservoir after a certain delay.

The third task is the NARMA task described in Jaeger (2003), Steil (2005) and others, where the following tenth-order system needs to be modeled: $y(k+1) = 0.3y(k) + 0.05y(k)[\sum_{i=0}^{9} y(k-i)] + 1.5u(k-9)u(k) + 0.1$. Here, the input to the reservoir $\mathbf{u}(k)$ is uniform random noise, and a single linear regression function is trained to reproduce $y(k+1)$.

2.1. Node complexity

When solving an engineering task using Reservoir Computing, one of the choices that needs to be made is the neuron model with which to build the reservoir. In the case of analog signals (as opposed to spike trains), reservoirs can be built from linear nodes without an activation function that are very fast to simulate but offer inferior computational power or sigmoidal neurons. In the case of the sigmoidal neurons, the choice exists between a linear classifier as readout function (as for ESNs) or a recurrent layer of sigmoidal neurons (as for the BPDC learning rule). One can also implement sigmoidal neurons that have a form of internal memory, in which case they are called analog integrator neurons (Jaeger, 2002). The reservoir dynamics are described in this case by

$$\mathbf{s}(k+1) = \left(1 - \frac{\delta}{\tau_{\text{int}}}\right) \cdot \mathbf{s}(k) + \frac{\delta}{\tau_{\text{int}}} \cdot \tanh(\mathbf{u}(k) \cdot \mathbf{W}_{\text{in}} + \mathbf{s}(k) \cdot \mathbf{W}), \quad (1)$$

where $\mathbf{s}$ is a vector containing the activation of the neurons in the reservoir, $\mathbf{W}_{\text{in}}$ and $\mathbf{W}$ are the input and reservoir weight matrices and $\mathbf{u}$ the input to the reservoir. Note that here the integration factor of $1/\tau_{\text{int}}$ is applied outside of the nonlinearity, in contrast to the reservoirs used by the BPDC learning rule. For the remainder of this contribution, we set the timestep δ to one for reasons of simplicity. The factor $1/\tau_{\text{int}}$ is a measure for the amount of internal memory of the neurons: if τ_{int} is equal to one, the nodes have no internal memory and behave as classic sigmoid neurons. For integration timeconstants τ_{int} larger than one, the self-feedback is enhanced. For clarity, we will refer to this parameter in the form of the retainment rate $r = 1 - 1/\tau_{\text{int}}$, which can range between [0, 1] and is set to 0.2 for the experiments in this subsection. For a more elaborate discussion of this parameter, see Section 2.2.

When we move into the domain of the spiking neurons, one of the simplest models is the LIF neuron which consists of a leaky membrane whose potential is increased by incoming spikes and that emits a spike itself once the potential reaches a threshold. This model can be extended by modeling the function of exponentially decaying synapses. For this contribution, we used a simplified synapse model called the Booij model that allows fast and accurate simulation. However, to make the comparison with LIF neurons without a synapse model correct, we needed to rescale the weight matrix (this is further detailed in Appendix B).

Besides the nodetype, two major parameters that determine the reservoir dynamics and performance are its size and its spectral radius. The spectral radius is the largest absolute value of the eigenvalues of the weight matrix and for discrete linear time-invariant systems, a spectral radius smaller than one guarantees asymptotic stability,[2] which means that the dynamics caused by an input pulse eventually die out. It is introduced in Jaeger (2001a) as a loose bound for the

[2] If the spectral radius is exactly one, the system is marginally stable.

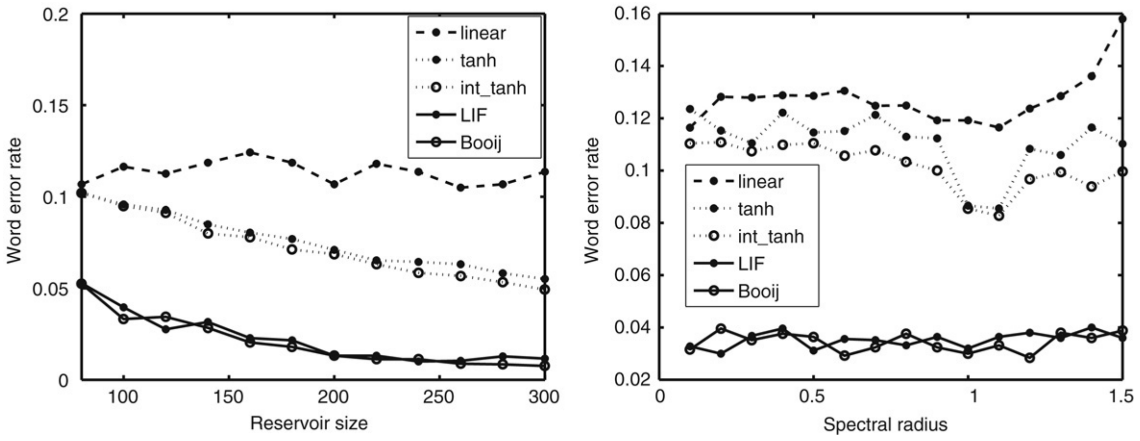

Fig. 1. These figures show the influence of the node complexity on the performance for the speech recognition task, for linear, tanh, analog integrator, LIF and Booij nodes, on the left as a function of reservoir size, and on the right as a function of the spectral radius of the connection matrix.

echo state property of sigmoidal reservoirs. The idea of the echo state property is – informally – that the reservoir needs to exhibit dynamics between unstable behaviour (where the network dynamics drown out the information contained in the inputs) and too stable dynamics that retain no information about the inputs. The extension of this measure to spiking neural networks loses some of its relevance, but it does offer a comparison between reservoirs built from analog and spiking neurons that use weight matrices with the same global scaling. Also, note that for the following discussion the spiking reservoirs were built in the same way as the analog neurons in order to allow a fair comparison — meaning that the reservoirs were built using a random connectivity with a connection fraction of 0.5. This method of building reservoirs should be contrasted with the 'LSM' topology, where typically three-dimensional (3D) neural microcolumns are built where the probability of two neurons being connected is related to the Euclidian distance between them. The latter reservoir topology is far more sparse than the former.

Fig. 1 shows the errors attained for the three benchmark problems described above as a function of reservoir size and spectral radius. For the figures on the left-hand side, reservoirs were created with a fixed spectral radius of 0.9 as per the guideline for ESNs, and for the figures on the right-hand side a reservoir size of 100 neurons was chosen. For the NARMA and memory capacity task, results for spiking reservoirs were omitted since these reservoirs are not suited for this problem because of the high-frequency nature of the signals, which cannot accurately be captured in spike trains using the filter-based coding scheme we used, and because the memory capacity measure cannot trivially be extended to spiking reservoirs, since it measures correlation between two continuous temporal signals. Also, results for BPDC-trained reservoirs are shown only for the NARMA task since the literature only describes timeseries prediction results and we were unable to get satisfactory results for the other problems.

For the speech task (Fig. 1), performance increases for larger reservoirs for all node types except linear, and memory limitations were reached before an optimum was found[3] — though we expect performance to eventually level out. Also, the temporal processing capabilities of spiking neurons offer a clear advantage, but the use of a synapse model does not offer any further improvement — perhaps this is because the timeconstants of the membrane and synapse are too similar to each other; see Appendix B. The figure on the right shows that, for sigmoidal neurons, the spectral radius is a significant parameter and the optimum is achieved for a spectral radius higher than one (around 1.1) (which is contrary to the claims in Jaeger (2002) that an optimal spectral radius lies close to one) but for spiking neurons it has no influence at all. This is probably due to the fact that the reservoir dynamics for spiking nodes are more controlled by the nonlinear effects and temporal integration properties of the spiking neurons, rather than the global scaling of the weight matrix.

For the memory capacity task (Fig. 2), the linear nodes show the highest memory capacity, which is an experimental confirmation of the theoretical result proved in Jaeger (2001b). Also, our results show that the memory capacity of analog integrator neurons is lower than that of sigmoidal neurons. This is caused by the fact that the integrating property of these reservoirs slows the dynamics down and thus the information on a very fast timescale contained in the input is lost. Thus, information on a slow timescale will be better preserved by analog integrator neurons, but quickly varying signals become muddled. Finally, the memory capacity of the reservoirs increases with size, since larger reservoirs will be able to store more information for a longer time due to the larger recurrent loops and higher dimensionality of the reservoir state. See Section 2.2 for more results on this.

Finally, the figures for the NARMA task (Fig. 3) indicate that reservoir size is not a very significant parameter for this task for any node type except for the tanh and integrator neurons where we see a slight increase in performance for larger reservoirs. Probably, the memory needed to accurately

[3] Note that we used a rather densely connected reservoir topology. Sparse topologies and corresponding matrices would allow larger reservoirs to be simulated.

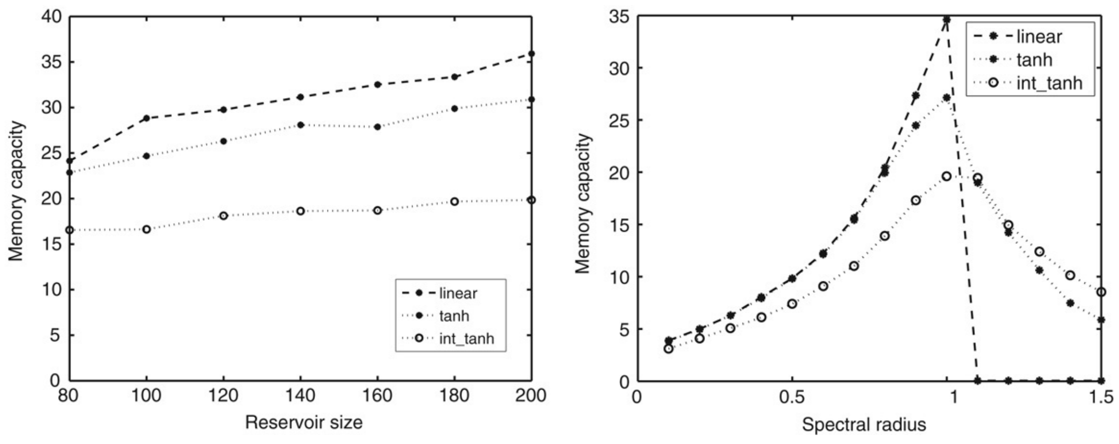

Fig. 2. These figures show the influence of the node complexity on the performance for the memory capacity task, for linear, tanh and analog integrator nodes, on the left as a function of reservoir size, and on the right as a function of the spectral radius of the connection matrix.

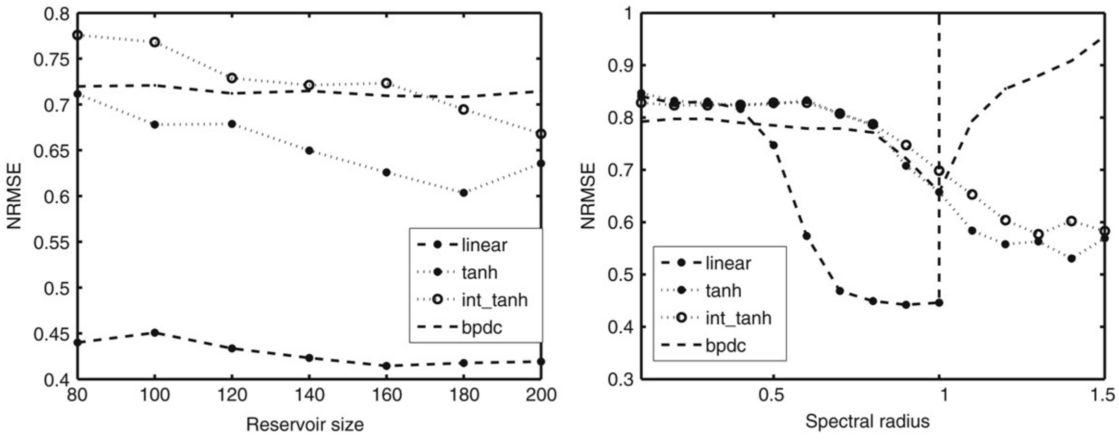

Fig. 3. These figures show the influence of the node complexity on the performance for the NARMA task, for linear, tanh, analog integrator nodes, and for the BPDC learning rule, on the left as a function of reservoir size, and on the right as a function of the spectral radius of the connection matrix.

model the tenth-order system is already present in very small reservoirs, and the only thing that causes the difference in performance between the nodetypes is the type of the nonlinear mapping. The significance of the spectral radius as an indicator for performance is quite obvious for this task (lower plot). For this task, we see that the guideline of an optimal spectral radius close to one is not optimal for all the nodes. We find that the BPDC rule performs worse overall than ESNs, but this could be due to the sensitivity of the learning rule to parameter settings and our limited experience with it. Note also that this figure shows that the spectral radius is only an approximate measure of stability for sigmoidal reservoirs: the performance of the linear nodes falls dramatically when the spectral radius is larger than one – indicating a drastic change in the reservoir dynamics – whereas for sigmoidal and analog integrator neurons this decrease is far more gradual.

2.2. *Node memory versus reservoir memory*

One of the key properties of reservoirs that sets them apart from more traditional approaches to classification or regression is their ability to retain information about the inputs for a certain time. This property is called the fading memory or echo state property in the context of LSMs and ESNs respectively, and is caused by the presence of recurrent connections inside the reservoir. Certain node types, however, possess some form of *internal* memory capability: the analog integrator neuron and both spiking neuron models (the LIF and Booij model).

When solving a task using reservoir computing techniques, the optimal value for the memory capability on both the node and the network level are determined by the timescales inherent to the problem. This means that a tradeoff needs to be made: one wants to have a reservoir that memorizes just enough information from the inputs, and this memory capability can be caused by the network size (larger networks have more memory) or the node memory. The NARMA task needs a memory of at least ten past timesteps (since the output is determined by input and outputs from up to ten timesteps ago), while the memory task needs more memory (up to 100 timesteps). Due to the complexity of the task, it is not easy to indicate the appropriate timescales for the speech recognition task (quite probably several timescales are at play there). In this section, we investigate the relation between both types of memory for the three benchmarks by using reservoirs built from analog integrator neurons and varying both reservoir size and the node memory.

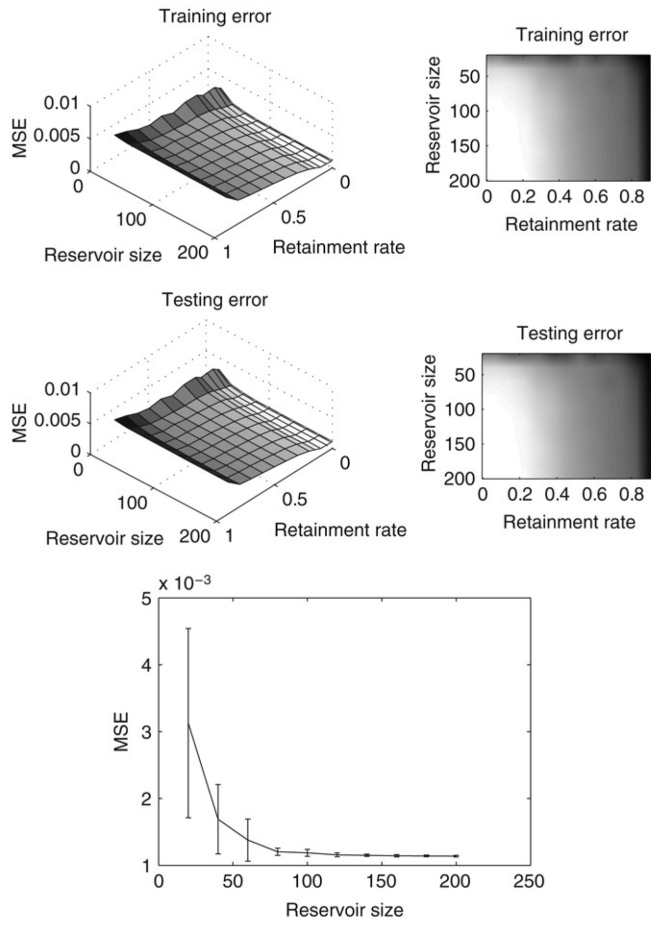

Fig. 4. Top plots show the reservoir size versus retainment rate r for the NARMA task. The bottom plot shows the variance of the performance for retainment rate $r = 0$ for the NARMA task.

Fig. 4 shows the results for the NARMA benchmark as a function of both reservoir size and the retainment rate r. These figures confirm the statement made earlier that the retainment rate has a negative effect on the performance of larger reservoirs for this task since it slows down the inherent timescale of the reservoir, and for this task – especially since the input is a random signal – rather fast dynamics are optimal. We also see the same independence of performance in relation to the reservoir size for retainment rates larger than zero, but for r approaching zero, there appears a decrease in performance for very small networks — a phenomenon that is not present for larger retainment rates (close to one). The lower plot of Fig. 4 shows the variance on the performance for the reservoirs with a retainment rate of zero. Here we see that the variance is far larger for small reservoirs — this means that, for larger reservoirs, the readout is far more likely to be able to extract the relevant information from the reservoir dynamics, whereas for small reservoirs good candidates exist but many more perform badly. Thus, we conclude that for very small reservoirs and no node integration the memory caused by the reservoir size becomes insufficient to retain enough information about past inputs. However, this effect cannot be detected for higher retainment rates since the reservoir states are mainly driven by the internal network dynamics instead of the external input.

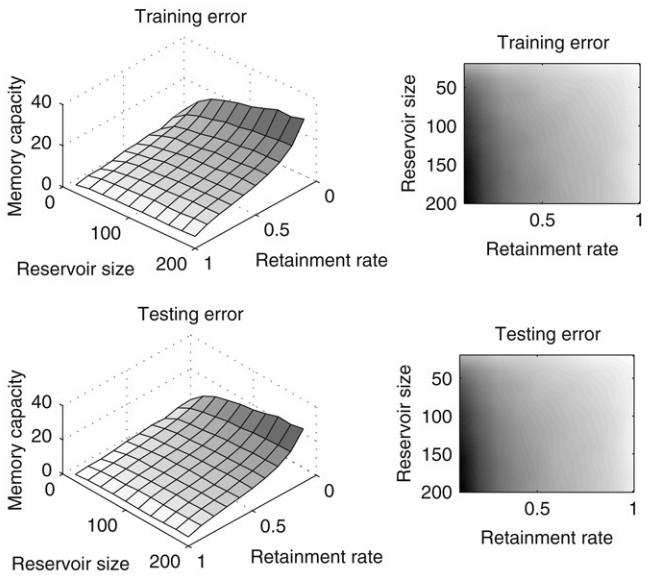

Fig. 5. Reservoir size versus retainment rate r for the memorization task.

The figure for the memory capacity as a function of reservoir size and retainment rate (Fig. 5) indicates as expected that larger reservoirs have a larger memory capacity. Also, as for the NARMA task, the information timescale present in the input signal that is relevant for the output is very quick since larger retainment rates cause a decrease of the memory capacity. Also, even though the landscape is more monotonic than for the NARMA task, one can see the same 'attenuating' effect of large retainment rates on the curves for reservoir size: the increase in memory capacity for larger reservoirs is far steeper for a retainment rate of zero than for r close to one.

Finally, Fig. 6 shows the results for the speech recognition task. Here, the influence of the reservoir size is more prominent than in the previous cases (note the different orientation of the error surface compared to the NARMA task). Also, the influence of the retainment rate is quite different compared to the previous two tasks. Here we see that quite large retainment rates are optimal — indicating the fact that as expected far slower input timescales are relevant for the performance of the classifier. These figures indicate that a very 'slow' or inert reservoir is beneficial for the performance, which might be expected considering the rather slow rate at which speech features change in relation to the sampling frequency.

2.3. Linking different bounds for the echo state property to network dynamics

In Jaeger (2002), two bounds are described for a reservoir built from sigmoidal neurons to have the echo state property. The first bound is based on the spectral radius ρ of the connection matrix and offers a necessary condition for the echo state property, namely that $\rho < 1$, meaning that all eigenvalues lie within the unit circle on the complex plane. This condition expresses that the reservoir is locally asymptotically stable around the origin. The second bound uses the largest singular value $\overline{\sigma}$, and expresses a the sufficient condition for the echo state property as $\overline{\sigma} < 1$. The spectral radius ρ is used in Jaeger

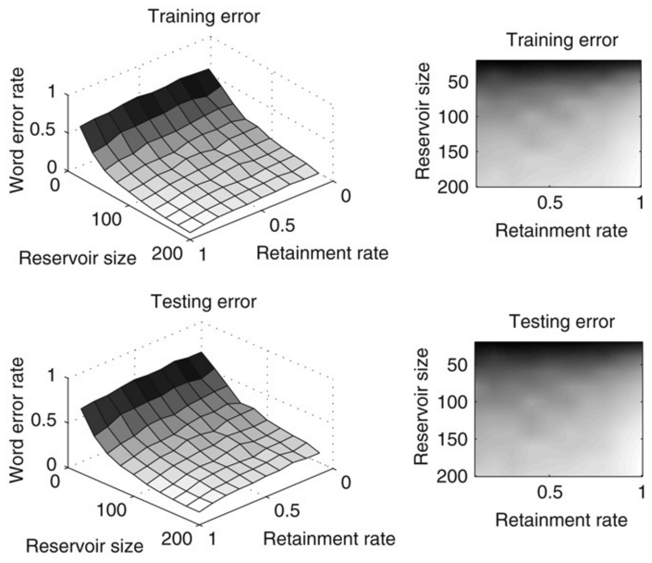

Fig. 6. Reservoir size versus retainment rate r for the speech task.

(2002) to suggest a heuristic method of constructing reservoirs: the idea is to start with a randomly connected network and to rescale the weights so that ρ is close to, but smaller than, one. Later, in Buehner and Young (2006) a tighter bound was proposed for constructing reservoirs, which is inspired by ideas from robust control theory. This condition uses the *structured singular value* μ, and expresses the fact that, if $\mu < 1$, the reservoir is globally asymptotically stable. This bound for the echo state property is tighter than the spectral radius, and for some specific weight matrices it is even an exact bound (i.e. the condition is both necessary and sufficient).

The bounds described above supply a measure for the computational quality of a reservoir and can all be deduced from the weight matrix of the reservoir, i.e. without simulating it explicitly on some input signals. However, it is not clear how these stability bounds relate to the actual network dynamics. In this context, the idea of '*computation at the edge of chaos*' is a recurring topic in literature on Reservoir Computing. Results from LSMs seem to indicate an optimal computational performance for reservoirs operating in a regime that lies between stable and chaotic behaviour (Legenstein & Maass, 2005), while Jaeger suggests that ESNs operate optimally in the stable regime but close to the border of instability (Jaeger, 2002), which is linked to the echo state property in the sense that a reservoir should exhibit sufficiently rich dynamics without drowning the information in chaotic behaviour.

One possible way of measuring the stability (or chaoticity) of an orbit through state space of a dynamic system is the Lyapunov exponent. The Lyapunov exponent is a measure for the exponential deviation in a certain direction of a trajectory, resulting from an infinitesimal disturbance in the state of the system. If the orbit is near an attractor, the effect of the disturbance will disappear and the Lyapunov exponent will be smaller than zero. In case the system drifts away from the orbit exponentially, the Lyapunov exponent is larger than zero and the orbit is unstable. Note that the Lyapunov exponent can be positive in one direction and negative in another.

A common definition of a *chaotic orbit* is then that if (i) the orbit is not asymptotically periodic, (ii) no Lyapunov exponent is exactly zero and (iii) the largest Lyapunov exponent is larger than zero, the orbit is chaotic, meaning that the orbit deviates exponentially in at least one dimension.

The kth Lyapunov *number* L_k is related to the kth Lyapunov exponent h_k through $L_k = \exp(h_k)$. Thus, if a trajectory is chaotic, at least one Lyapunov number will be greater than one. For every trajectory $\mathbf{s}(t), t = 1, \ldots, n$ of a dynamical system in state space, we can calculate $k = 1, \ldots, m$ Lyapunov numbers (one for every dimension of the state) by considering a hypersphere in state space whose center follows the trajectory $\mathbf{s}(t)$, and considering how it is transformed by the update function of the system. The hypersphere will evolve to a hyper-ellipsoid as it travels through state space. The Lyapunov numbers for the trajectory are then given by

$$L_k = \lim_{n\to\infty} (r_k^n)^{1/n},$$

where r_k^n is the length of kth longest orthogonal axis of the hyper-ellipsoid at discrete timestep n. The value of these lengths can be calculated as follows. Let J_n denote the Jacobian of the nth iterative application of the map $\mathbf{f}$. The length of the axes of the hyper-ellipsoid is then given by the square root of the eigenvalues of $J_n J_n^{\mathrm{T}}$ (see Alligood, Sauer, and Yorke (1996) for a more elaborate discussion). Thus, the Lyapunov number L_k expresses the exponential expansion ($L_k > 1$) or contraction ($L_k < 1$) of a unit sphere under the map $\mathbf{f}$. Note that this method assumes that the limit above converges.

The Lyapunov definition given above is only suited for autonomous systems (or driven systems where the driving force can also be modeled as state variables, rendering the system autonomous). But in the case of reservoir computing we have systems that are constantly driven by complex, even random, input. The standard definition of a Lyapunov exponent as the existence of a limit will therefore not be applicable because no steady state is reached due to the continuously changing input and state trajectory.

In Legenstein and Maass (2005), the Lyapunov exponent is measured empirically for a spiking reservoir by calculating the average Euclidian distance of the reservoir states resulting from time-shifting a single input spike over 0.5 ms. To our knowledge, this is the only investigation of the reservoir dynamics using the Lyapunov criterion, and for spiking neurons this experimental approach is the only way to formulate any conclusions about the stability of the system due to the complexity and time-dependence of the model. This method can be seen as the disturbance of an autonomous system where the exponential state deviation is an approximation of the Lyapunov exponent. A considerable disadvantage of this method is that it is very time-consuming.

In the case of sigmoidal neurons, however, we can calculate a measure analytically that is closely related to the Lyapunov exponent, due to the simpler neuron model. The Jacobian matrix of $\mathbf{f(s)}$ at a point $\mathbf{s} \in \mathbb{R}^n$ is given by

$$\mathbf{Df(s)} = \begin{pmatrix} \frac{\partial f_1}{\partial s_1}(\mathbf{s}) & \cdots & \frac{\partial f_1}{\partial s_n}(\mathbf{s}) \\ \frac{\partial f_n}{\partial s_1}(\mathbf{s}) & \cdots & \frac{\partial f_n}{\partial s_n}(\mathbf{s}). \end{pmatrix}$$

A generic sigmoidal reservoir is described by the following map:

$$\begin{aligned}\mathbf{s}(k+1) &= \mathbf{f}(\mathbf{s}(k)) = \tanh(\mathbf{s}(k) * \mathbf{W}) \\ &= \left[\tanh\left(\sum_{i=1}^{n} w_{1i}\mathbf{s}_1\right) \quad \cdots \quad \tanh\left(\sum_{i=1}^{n} w_{ni}\mathbf{s}_n\right) \right].\end{aligned}$$

When we compute the Jacobian of the above map $\mathbf{f}(\mathbf{s}(k))$, this gives

$$\mathbf{Df}(\mathbf{s}(k)) = \begin{bmatrix} \left[1-\tanh^2\left(\sum_{i=1}^{n} w_{1i}s_1\right)\right]w_{11} & \cdots & \left[1-\tanh^2\left(\sum_{i=1}^{n} w_{1i}s_1\right)\right]w_{1n} \\ \vdots & & \vdots \\ \left[1-\tanh^2\left(\sum_{i=1}^{n} w_{ni}s_n\right)\right]w_{n1} & \cdots & \left[1-\tanh^2\left(\sum_{i=1}^{n} w_{ni}\mathbf{s}_n\right)\right]w_{nn} \end{bmatrix} \tag{2}$$

$$= \begin{bmatrix} \left[1-s_1^2(k)\right]w_{11} & \cdots & \left[1-s_1^2(k)\right]w_{1n} \\ \vdots & & \vdots \\ \left[1-s_n^2(k)\right]w_{n1} & \cdots & \left[1-s_n^2(k)\right]w_{nn}. \end{bmatrix}. \tag{3}$$

The above matrix offers an analytical expression for the Jacobian matrix J, which in turn allows us to approximate the Lyapunov numbers. It is indeed an approximation, because in this case we are dealing with an input driven system and we cannot compute the limit for $k \to \infty$. We will therefore refer to this measure as the pseudo-Lyapunov exponent $\tilde{h}_k$. We calculate the largest pseudo-Lyapunov exponent of a trajectory of N timesteps as

$$\tilde{h}_{\max} = \max_k \prod_{n=1}^{N} (r_k)^{1/n}, \tag{4}$$

where $r_k = \sqrt{|\lambda_k|}$, λ_k being the kth eigenvalue of $J_n J_n^{\mathrm{T}}$.

Due to the analytical derivation of this rule, it can easily be calculated, and is exact. An extra speed-up can be attained by calculating the pseudo-exponents in a sampled manner, in our case every ten timesteps. This proved to be an accurate approximation. We will demonstrate that this measure is an accurate indicator of reservoir dynamics and performance. Also, to our knowledge, this is the only analytically computed measure of reservoir *and* task-dependent dynamics.

To illustrate the relationship between these different bounds and reservoir performance for certain tasks, we evaluated the performance of different reservoirs on the NARMA (Fig. 7), memory (Fig. 8) and speech recognition benchmark (Fig. 9) by constructing the weight matrix of the reservoirs and then rescaling it so that either the spectral radius, LSV or μ has a specified value.[4] For each of these reservoirs, the largest

[4] Note that these measures are not linearly related, meaning they are not simply a rescaled version of each other. For a given value of one measure, the other two vary quite substantially.

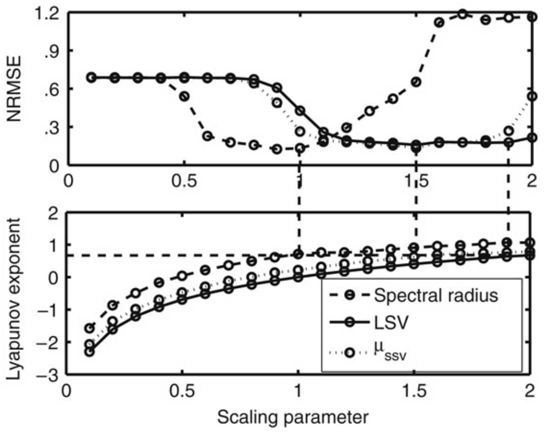

Fig. 7. The top figure shows the performance on the NARMA task as a function of different reservoir metrics. The bottom figure shows the corresponding pseudo-Lyapunov exponents.

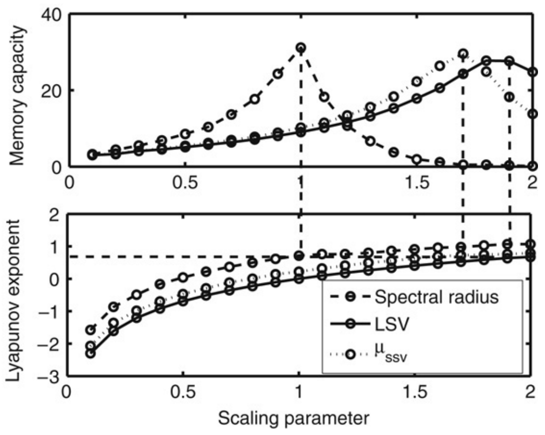

Fig. 8. The top figure shows the performance on the memory capacity task as a function of different reservoir metrics. The bottom figure shows the corresponding pseudo-Lyapunov exponents.

pseudo-Lyapunov exponent $\tilde{h}_{\max}$ was also measured for every orbit and averaged across all input timeseries.

In all three cases, the optimal value for the μ parameter lies between that of the spectral radius and of the LSV (which was expected since the spectral radius and LSV are a lower and upper bound for the echo state property), but it is significantly higher than one — which indicates that the reservoir is not globally asymptotically stable. This is also confirmed when the corresponding pseudo-Lyapunov exponent is measured, indicated by the dashed lines on the figure; it appears that the system is on average state-expanding for the orbits caused by the inputs for this task and the pseudo-Lyapunov exponent for the optimal values of the different metrics is very similar for both benchmarks and lies around 0.7, which indicates a dynamic regime: on average the trajectory locally shows exponential deviation in at least one direction. Note that the $\tilde{h}_{\max}$-curves are very similar for the NARMA and memory capacity tasks, since in both cases the input to the reservoirs is a uniform random signal. However, the fact that the plots for the speech recognition task are very similar adds to the validity of the pseudo-Lyapunov exponent as a measure of input-dependent reservoir dynamics. Note also that a largest

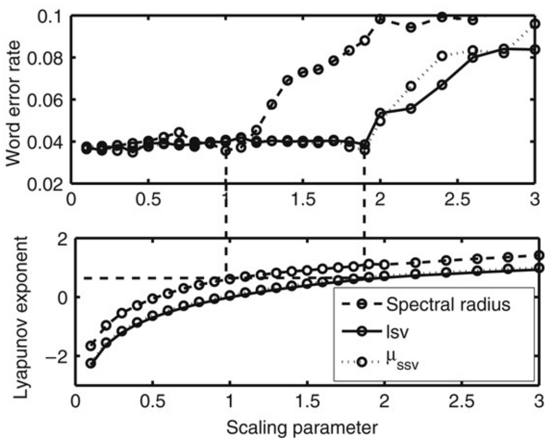

Fig. 9. The top figure shows the performance on the speech recognition task as a function of different reservoir metrics. The bottom figure shows the corresponding pseudo-Lyapunov exponents.

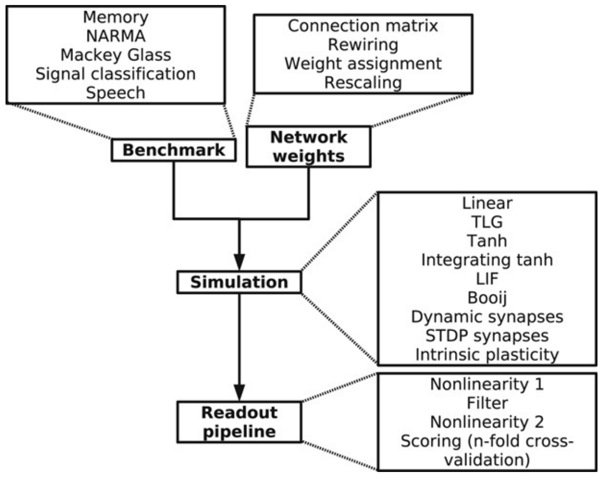

Fig. 10. Overview of the simulation pipeline for the RC toolbox.

pseudo-Lyapunov exponent larger than zero does not mean that a system is chaotic.

3. Reservoir computing toolbox

In this section, we will describe the RC toolbox (see Fig. 10) we have used for the work presented here, and which was developed at our laboratory. The toolbox is written in Matlab, is open source[5] and it offers a user-friendly interface for the simulation of all common reservoir computing implementations, such as the LSM, the ESN and the BPDC learning rule. Every parameter relevant to an experiment is set in a global configuration file, and *sweeps* across parameter ranges can be easily done. When multiple parameter ranges are specified, every possible combination will be simulated, which allows a detailed exploration of the performance for a certain benchmark in a certain area of the parameter space.

Usually, an experiment consists of a sweep across a certain parameter range, whereby every point in parameter space is simulated a number of times (*runs*) to capture the variance due to the stochastic construction of the networks. Every single run itself is set up as a modular pipeline, where every step can be configured and where new modules can easily be added (for instance a new benchmark test or reservoir construction algorithm). Plotting utilities are provided that yield the mean performance or error (according to the specified scoring function) and their standard deviation.[6] In the following subsections, we will introduce the different stages of the toolbox and discuss their possibilities.

3.1. Benchmark tests

The toolbox includes some of the benchmark tests that have been described in RC literature, including the Mackey–Glass timeseries prediction (Steil, 2005), the isolated spoken digit recognition task from Verstraeten et al. (2005), the NARMA task described in Jaeger (2003), Steil (2005) and others. This stage is easily extended to other datasets. Depending on the selected benchmark test, at least two input timeseries vectors and corresponding desired output vectors are generated. In case the reservoir is built from spiking neurons and the input is analog, a spike coding algorithm can be applied to convert the timeseries to spike trains. The toolbox offers three coding hypotheses: a simple rate coding scheme where the spike trains are generated by a Poisson process with the instantaneous firing rate equal to the analog value being encoded, a scheme that uses a Leaky Integrate and Fire neuron to perform the coding where the analog timeseries is fed into the neuron as a membrane current, and finally BSA (Schrauwen & Van Campenhout, 2003), a fast filter-based coding scheme that introduces a low amount of noise in the spiketrain compared to other methods.

3.2. Reservoir generation

Reservoirs are constructed in a stochastic manner, and the search for a method to construct a priori suitable reservoirs that are guaranteed or likely to offer a certain performance is the topic of much research (Jaeger, 2005). Several ad hoc methods that all yield satisfactory results have been proposed in the literature. For instance, Maass et al. (2002) construct their reservoirs in a biologically inspired way, whereby the nodes of the reservoir are placed on a 3D lattice (based on the *microcolumnar* structures found in the cortex), and the probability of a connection existing between two nodes a and b is given by $P_{\text{conn}}(a,b) = C \cdot \mathrm{e}^{\frac{-D^2(a,b)}{\lambda^2}}$, with $D(a,b)$ the distance between the nodes and C and λ parameters that control the reservoir connectivity and thus its dynamics. Several other construction heuristics for reservoir were proposed and tested by Jaeger, including random assignment of weights from a small set or a network with a certain fraction of nonzero, normally distributed weights. The toolbox offers a generic method of generating reservoirs that is split into several distinct steps.

[5] Available from http://www.elis.ugent.be/rct.

[6] This approach assumes that the scores obey a normal distribution; we performed several statistical tests that all confirm that this is the case.

- The connectivity is determined, i.e. which neurons are connected to each other. A wide range of possibilities is supplied, which includes a regular 2D or 3D lattice, with optional rewiring (Watts & Strogatz, 1998) (which yields small world networks[7] (Watts, 1999)) or a randomly connected network with a certain connection fraction.
- Weights are assigned to the connections according to some heuristic. For instance, they can be drawn from a normal distribution or taken from a discrete set of values.
- The global weight matrix can then be rescaled according to some measure, such as the spectral radius, largest single value or μ (ssv) value.

Using this generic reservoir generation method, all reservoir topologies currently described in the literature can be created as well as other novel topologies.

3.3. *Possible nodetypes*

Once the weight matrix of the network is determined, the reservoir can be simulated. The toolbox implements the simulation of a wide variety of neural models, ranging from very simple linear nodes, over analog integrator neurons up to spiking neural models. The simulation occurs either inside Matlab itself in the case of analog neurons, or by using a MEX interface to an external event simulator for spiking neural networks called ESSpiNN developed at our laboratory (D'Haene, Schrauwen, & Stroobandt, 2006). This simulator offers a rich subset of the neural models possible in CSIM (Natschläger, Markram, & Maass, 2002, chap. 9),[8] but it is event-based as opposed to timestep-based, which gives a much more efficient, fast and accurate simulation of spiking neural networks for simple neuron models. ESSpiNN can simulate Leaky Integrate and Fire neurons, optionally with one or multiple exponential synapse models (Gerstner & Kistler, 2002), intrinsic plasticity (Triesch, 2004), Spike Time Dependent Plasticity (Markram, Lübke, Frotscher, & Sakmann, 1997) or dynamic synapses (Markram, Pikus, Gupta, & Tsodyks, 1998). This means that all reservoir node types described in the literature are available for simulation.

3.4. *The output pipeline*

The reservoir response for each input timeseries is recorded during simulation and – in case the nodes are spiking neural models – low-pass filtered to mimic the operation of a neuron membrane (as described in Legenstein and Maass (2005)). This yields a time-dependent signal for every neuron in the reservoir, which can optionally be resampled at a lower rate `dt_resample`. The result is a large matrix of size $N \times T_{\text{rsmp}}$, with T_{rsmp} the number of timesteps the simulation has run for in units of `dt_resample`. This matrix is multiplied with the weight matrix of a linear or nonlinear regression function of size $N \times N_{\text{out}}$, where N_{out} is the number of classes in the case of classification, or the dimensionality of the output vector in the case of a regression task. This results in an $N_{\text{out}} \times T_{\text{rsmp}}$ matrix, which can then optionally be fed into a post-processing pipeline, consisting of a first nonlinearity, a filtering operation, and a second nonlinearity. These last three steps allow one to transform the signals spatially (using the nonlinearities) or temporally (using the filter). Examples for the nonlinearity operation include the identity function (i.e. no nonlinear operation), the tanh() function, the sign function or winner-take-all across all timesteps. The filtering operation could consist of a low-pass filter with a given timeconstant, or a simple mean operation across all timesteps. In the case of the speech recognition task, for instance, the post-processing pipeline consists of a mean operation followed by winner-take-all. The modular structure of the post-processing pipeline allows the easy addition of other functions.

The BPDC learning rule is also implemented in the toolbox, and all benchmark and reservoir generation functions can be used with this learning rule.

4. Conclusions and future work

In this contribution, we have presented an experimental investigation of the different node types currently described, for a broad range of parameter settings and for a variety of tasks. It appears that, for the speech task, the use of spiking reservoirs is quite beneficial, and that the reservoir size has a significant influence on the performance. Also, for this task, the optimal value for the spectral radius is larger than one, contrary to previous results, and this parameter has no influence on performance for spiking reservoirs. For the memorization task, we found a monotonic increase of the memory capacity as a function of the reservoir size, and also a negative effect of a retainment rate larger than zero. We have found a strong dependence on the spectral radius for analog neurons and an optimum for a spectral radius of one. Similar results were found for the NARMA task. Further, we investigated the influence of the memory function within the nodes and within the reservoir on the reservoir performance and have illustrated the tradeoff between network and node memory for three tasks with very different memory requirements.

We also empirically investigated the relation between several bounds for the echo state property described in the literature and the network dynamics by analytically calculating an approximation of the Lyapunov exponent. This pseudo-Lyapunov exponent is an accurate measure of the task-dependent network dynamics and also a very good indicator of the performance on all of the benchmark tests. We investigated the link between the pseudo-Lyapunov exponents and static bounds based strictly on the weight matrix of the reservoir, and found that optimal values of these bounds correspond to the same exponent value.

Finally, we described the toolbox we used for this work, that offers all current implementations of reservoir computing described in the literature, and that extends this spectrum

[7] Networks with high local and low global connectivity — this connectivity is found in biological neural networks, but also in social, electrical and many other networks.

[8] Only complex biological models such as the Hodgkin–Huxley model are absent because these cannot be simulated in an event-based manner.

with additional benchmark problems, network generation algorithms, neural models and post-processing methods. This toolbox allows the user-friendly simulation of a broad range of reservoir types and topologies, the easy optimization of network parameters and a structured way of storing and plotting simulation results.

Future research work includes a thorough investigation of the noise robustness of different neural models — one can hypothesize that spiking neural models are more noise robust due to their inherent filtering behaviour. Also, research into the performance of novel network topologies (such as scale-free networks) is a potential direction for future work, since the network dynamics are largely determined by the connection topology. Finally, we plan to extend the research of the pseudo-Lyapunov exponent as a measure of network dynamics to other benchmark types and reservoir topologies.

Acknowledgements

David Verstraeten and Michiel D'Haene are sponsored by the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen). Benjamin Schrauwen is sponsored by the FWO Flanders project G.0317.05.

Appendix A. Specification of experiments

All reservoirs used for experiments in this contribution were constructed as follows: a random matrix was generated whereby a fraction of the entries is set to one (the *connection fraction*). In our experiments, we used a connection fraction of 0.5. Next, the nonzero entries are assigned random values from a normal distribution. Finally, the weight matrix is rescaled to set the spectral radius, LSV or μ to a certain value. The input matrix is constructed in the same way, with a connection fraction of 0.1, but is left unscaled.

A.1. NARMA task

As cited in the main text, the Non-Linear Auto-Regressive Moving Average (NARMA) task consists of modeling the output of the following tenth-order system: $y(k+1) = 0.3y(k) + 0.05y(k)[\sum_{i=0}^{9} y(k-i)] + 1.5u(k-9)u(k) + 0.1$. Here, $u(k)$ is a uniform random signal in $[0, 1]$, which serves as the sole input to the reservoir. The readout is trained to reproduce the signal $y(k+1)$. The performance is measured as the mean square error (MSE), defined as $\text{MSE} = \frac{1}{n}\sum_{t=1}^{n}(s(t) - y(t))^2$, with $s(t)$ the readout output and $y(t)$ the desired output signal. For each reservoir that was constructed, a readout was trained on an example output of 1000 timesteps, and tested on a different input signal of the same length.

A.2. Memorization task

This task was introduced in Jaeger (2002). Here, the input signal is again a random uniform signal, in the interval $[-0.8, 0.8]$. The desired output is multi-dimensional, and consists of a series of delayed versions of the input signal. In this case, a series of 100 regressors was trained to reproduce the input signal delayed over 1 up to 100 timesteps. The performance here is the memory capacity (MC), defined in Jaeger (2002) as $\text{MC} = \sum_{k=1}^{100} \text{MC}_k$, with the k-delay memory capacity $\text{MC}_k = \max_{W_k} d[W_k](s_k(t), y_k(t)) = \max_{W_k} \frac{\text{cov}^2(s_k(t), y_k(t))}{\sigma^2(s_k(t))\sigma^2(y_k(t))}$, with $s_k(t)$ the output of the kth regressor and $y_k(t)$ the input signal delayed over k timesteps. Here, $d[W_k]$ denotes the determination coefficient for the kth readout weight matrix, which is a measure of the variance in one signal caused by the other. The maximum determination coefficient is automatically achieved by training the readout weights using the pseudo-inverse or Moore–Penrose inverse method, as is done here.

A.3. Speech recognition task

The speech recognition task described here consists of the recognition of isolated digits. The dataset is a subset of the TI46 speech corpus, and consists of ten digits, zero to nine, each uttered ten times by five different female speakers, resulting in 500 speech fragments sampled at 12 kHz. The speech was preprocessed using the so-called Lyon Passive Ear model of the human cochlea (Lyon, 1982) using Malcolm Slaney's Auditory Toolbox for Matlab (Slaney, 1988). This inner ear model is built in three stages: a band-pass filterbank that reflects the human selectivity for certain frequencies, followed by half-wave rectification and automatic gain control (AGC) to model the nonlinear properties of the haircells in the cochlea. The number of filters in the filterbank determines the dimensionality of the model output — in this case there are 77 channels. The result of this preprocessing step is either directly fed into the reservoir in the case of non-spiking nodes, or else transformed into spike trains using a method called BSA (Ben's Spiker Algorithm) (Schrauwen & Van Campenhout, 2003). This algorithm takes a temporal signal and a filter as input, and yields the spiketrain that will optimally reconstruct the original signal when decoded using this filter. The algorithm works by scanning across the signal, and at each timestep computing a heuristic error measure. When this error measure exceeds a threshold, a spike is generated and the filter is subtracted from the signal. In this case, the filter is exponentially decaying with a timeconstant of 30 sample timesteps.

In the case of spiking neurons, the spike trains generated by the reservoir are transformed back into the analog domain by filtering them with an exponential filter, which mimics the internal operation of the membrane potential of a simple spiking neuron. These analog signals are then subsampled by a factor of 20.

Ten different linear classifiers are trained, each sensitive to a different digit in the vocabulary. These classifiers are trained to output the correct value at each timestep. When tested, a final classification is reached by taking the temporal mean of the output of every classifier, and applying winner-take-all to the resulting vector. The total performance across the dataset is computed using ten-fold cross-validation.

Appendix B. Fast simulation of Leaky Integrate and Fire neurons

Event-based simulation of LIF neurons requires the calculation of the firing times of a neuron, i.e. the time when the membrane potential $u_i(t)$ crosses the threshold value θ. In the general case, every incoming connection has a separate synapse, each with its own synaptic timeconstant τ_s. The membrane potential is then given by

$$u_i(t) = \int_0^\infty \mathrm{e}^{\left(-\frac{s}{\tau_m}\right)} \left[\sum_j w_{ij} \sum_f \alpha(t - t_j^{(f)} - s) - (\theta - u_r) \sum_f \delta(t - t_i^{(f)} - s) \right] \mathrm{d}s \tag{B.1}$$

which needs to be solved for t. This is generally a transcendent equation that needs to be approximated numerically using a root-finding algorithm such as Newton–Raphson. However, when we assume that every incoming connection to a neuron has the same synaptic timeconstant τ_s and that it is half the membrane timeconstant of the neuron $\tau_s = \frac{\tau_m}{2}$, then this transcendent equation becomes a simple quadratic equation that can be solved analytically and very quickly. This simplification was coined by Olaf Booij.[9]

However, when we construct reservoirs consisting of these LIF neurons with a simplified synapse model, we need to take these limitations into account. Indeed, without further precautions, a reservoir with the same weight matrix will show far less activity when simulated with this simplified model due to the presence of a synapse model. For the same input spiketrain, the total current flowing to the neuron membrane will be less in case synapses are modeled, which results in a lower number of emitted spikes. To compensate for this, the weight matrix needs to be scaled by an appropriate factor. This factor can be determined as follows.

When we hypothesize that there is a high activity inside the reservoir,[10] we can find the rescaling factor by demanding that the integral of the post-synaptic potential (PSP) over time of both the LIF without synapse model and the simplified synapse model be equal. The integral over time for the PSP of an LIF neuron without synapse is given by

$$\int_0^\infty \exp\left(-\frac{t}{\tau_m}\right) \mathrm{d}t = \tau_m, \tag{B.2}$$

while the same quantity for an LIF neuron with a simplified synapse model is given by

$$\int_0^\infty \exp\left(-\frac{t}{\tau_m}\right) - \exp\left(-\frac{t}{\tau_s}\right) \mathrm{d}t = \tau_m - \tau_s. \tag{B.3}$$

In the case of the Booij model, $\tau_s = \frac{\tau_m}{2}$, so we find that we need to scale the weights by a factor of 2 in order to achieve the same global activity. This approximation turns out to be accurate for a very broad range of firing rates down to 10 spikes/s.

[9] Personal communication.

[10] Our experience is that this is generally the case for spiking reservoirs.

References

Alligood, K., Sauer, T., & Yorke, J. (1996). *Chaos: An introduction to dynamical systems*. Springer.

Atiya, A. F., & Parlos, A. G. (2000). New results on recurrent network training: Unifying the algorithms and accelerating convergence. *IEEE Transactions on Neural Networks*, *11*, 697.

Buehner, M., & Young, P. (2006). A tighter bound for the echo state property. *IEEE Transactions on Neural Networks*, *17*(3), 820–824.

Burgsteiner, H. (2005a). On learning with recurrent spiking neural networks and their applications to robot control with real-world devices. *Doctoral dissertation*. Graz University of Technology.

Burgsteiner, H. (2005b). Training networks of biological realistic spiking neurons for real-time robot control. In *Proceedings of the 9th international conference on engineering applications of neural networks* (pp. 129–136).

Bush, K., & Anderson, C. (2005). Modeling reward functions for incomplete state representations via echo state networks. In *Proceedings of the international joint conference on neural networks* (pp. 2995–3000).

D'Haene, M., Schrauwen, B., & Stroobandt, D. ((2006/9)). Accelerating event based simulation for multi-synapse spiking neural networks. In *Proceedings of the 16th international conference on artificial neural networks: Vol. 4131* (pp. 760–769). Athens, Berlin, Heidelberg: Springer.

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

Gers, F., & Schmidhuber, J. (2001). LSTM recurrent networks learn simple context free and context sensitive languages. *IEEE Transactions on Neural Networks*, *12*(6), 1333–1340.

Gerstner, W., & Kistler, W. M. (2002). *Spiking neuron models*. Cambridge University Press.

Graves, A., Eck, D., Beringer, N., & Schmidhuber, J. (2004). Biologically plausible speech recognition with LSTM neural nets. In *Proceedings of BIO-ADIT* (pp. 127–136).

Haykin, S. (1999). *Neural networks: A comprehensive foundation* (2nd ed.). Prentice Hall.

Jaeger, H. (2001a). The "echo state" approach to analysing and training recurrent neural networks. *Tech. rep. no. GMD report 148*. German National Research Center for Information Technology.

Jaeger, H. (2001b). Short term memory in echo state networks. *Tech. rep. no. GMD report 152*. German National Research Center for Information Technology.

Jaeger, H. (2002). Tutorial on training recurrent neural networks, covering BPTT, RTRL, EKF and the "echo state network" approach. *Tech. rep. no. GMD report 159*. German National Research Center for Information Technology.

Jaeger, H. (2003). Adaptive nonlinear system identification with echo state networks. In *Advances in neural information processing systems* (pp. 593–600).

Jaeger, H. (2005). Reservoir riddles: Suggestions for echo state network research (extended abstract). In *Proceedings of the international joint conference on neural networks* (pp. 1460–1462).

Jaeger, H., & Haas, H. (2004). Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless telecommunication. *Science*, *308*, 78–80.

Joshi, P., & Maass, W. (2004). Movement generation and control with generic neural microcircuits. In *Proceedings of BIO-ADIT* (pp. 16–31).

Kilian, J., & Siegelmann, H. (1996). The dynamic universality of sigmoidal neural networks. *Information and Computation*, *128*, 48–56.

Legenstein, R., & Maass, W. (2005). What makes a dynamical system computationally powerful? In S. Haykin, J. Principe, T. Sejnowski, & J. McWhirter (Eds.), *New directions in statistical signal processing: From systems to brain*. MIT Press.

Lyon, R. (1982). A computational model of filtering, detection and compression in the cochlea. In *Proceedings of the IEEE ICASSP* (pp. 1282–1285).

Maass, W. (1997). Noisy spiking neurons with temporal coding have more computational power than sigmoidal neurons. In M. Mozer, M. I. Jordan, & T. Petsche (Eds.), *Advances in neural information processing systems*: *Vol. 9* (pp. 211–217). Cambridge: MIT Press.

Maass, W., & Bishop, C. (2001). *Pulsed neural networks*. Cambridge, MA: Bradford Books/MIT Press.

Maass, W., Legenstein, R.A., & Markram, H. (2002). A new approach towards vision suggested by biologically realistic neural microcircuit models. In *Lecture notes in computer science*: *Vol. 2525* (pp. 282–293). *Proceedings of the 2nd workshop on biologically motivated computer vision*. Berlin: Springer.

Maass, W., Natschläger, T., & Markram, H. (2002). Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, *14*(11), 2531–2560.

Maass, W., Natschläger, T., & Markram, H. (2003). A model for real-time computation in generic neural microcircuits. In *Proceedings of NIPS*: *Vol. 15* (pp. 229–236). MIT Press.

Maass, W., Natschläger, T., & Markram, H. (2004). Fading memory and kernel properties of generic cortical microcircuit models. *Journal of Physiology*, *98*(4–6), 315–330.

Markram, H., Lübke, J., Frotscher, M., & Sakmann, B. (1997). Regulation of synaptic efficacy by coincidence of postsynaptic APS and EPSPS. *Science*, *275*(5297), 213–215.

Markram, H., Pikus, D., Gupta, A., & Tsodyks, M. (1998). Potential for multiple mechanisms, phenomena and algorithms for synaptic plasticity at single synapses. *Neuropharmacology*, *37*, 489–500.

Minsky, M., & Papert, S. (1969). *Perceptrons: An introduction to computational geometry*. Cambridge, MA: MIT Press.

Natschläger, T., Bertschinger, N., & Legenstein, R. (2004). At the edge of chaos: Real-time computations and self-organized criticality in recurrent neural networks. In *Proceedings of NIPS '04* (pp. 145–152).

Natschläger, T., Markram, H., & Maass, W. (2002). Computer models and analysis tools for neural microcircuits. In R. Kötter (Ed.), *A practical guide to neuroscience databases and associated tools*. Boston: Kluwer Academic Publishers.

Omlin, C. W., & Giles, C. L. (1994). Constructing deterministic finite-state automata in sparse recurrent neural networks. In *IEEE international conference on neural networks* (pp. 1732–1737). Piscataway, NJ: IEEE Press.

Robinson, A. J. (1994). An application of recurrent nets to phone probability estimation. *IEEE Transactions on Neural Networks*, *5*, 298–305.

Rodriguez, P. (2001). Simple recurrent networks learn context-free and context-sensitive languages by counting. *Neural Computation*, *13*(9), 2093–2118.

Schrauwen, B., & Van Campenhout, J. (2003). BSA, a fast and accurate spike train encoding scheme. In *Proceedings of IJCNN* (pp. 2825–2830).

Skowronski, M. D., & Harris, J. G. (2006). Minimum mean squared error time series classification using an echo state network prediction model. In *IEEE international symposium on circuits and systems*.

Slaney, M. (1988). Lyon's cochlear model. *Tech. rep. no. 13*. Advanced Technology Group, Apple Computer Inc.

Steil, J. (2005). Memory in backpropagation–decorrelation: O(N) efficient online recurrent learning. In *Proceedings of the international conference on artificial neural networks* (pp. 649–654).

Steil, J. (2006). Online stability of backpropagation–decorrelation recurrent learning. *Neurocomputing*, *69*, 642–650.

Steil, J. J. (2004). Backpropagation–Decorrelation: Online recurrent learning with O(N) complexity. In *Proceedings of IJCNN '04*: *Vol. 1* (pp. 843–848).

Suykens, J., Vandewalle, J., & De Moor, B. (1996). *Artificial neural networks for modeling and control of non-linear systems*. Springer.

Suykens, J., & Vandewalle, J. (Eds.) (1998). *Nonlinear modeling: Advanced black-box techniques*, (pp. 29–53). Kluwer Academic Publishers.

Triesch, J. (2004). Synergies between intrinsic and synaptic plasticity in individual model neurons. In *Neural information processing systems*: *Vol. 17*.

Vapnik, V. (1995). *The nature of statistical learning theory*. New York: Springer-Verlag.

Verstraeten, D., Schrauwen, B., & Stroobandt, D. (2005/11). Reservoir computing with stochastic bitstream neurons. In *Proceedings of the 16th annual Prorisc workshop* (pp. 454–459).

Verstraeten, D., Schrauwen, B., Stroobandt, D., & Van Campenhout, J. (2005). Isolated word recognition with the liquid state machine: A case study. *Information Processing Letters*, *95*(6), 521–528.

Watts, D., & Strogatz, S. H. (1998). Collective dynamics of 'small-world' networks. *Nature*, *39*, 440–442.

Watts, D. J. (1999). *Small worlds: The dynamics of networks between order and randomness*. Princeton University Press.