Optoelectronic Reservoir Computing: Supplementary Material

Y. Paquot[1], F. Duport[1], A. Smerieri[1], J. Dambre[2], B. Schrauwen[2], M. Haelterman[1], S. Massar[3*]

[1]Service OPERA-Photonique, Université libre de Bruxelles (U.L.B.), 50 Avenue F. D. Roosevelt, CP 194/5, B-1050 Bruxelles, Belgium

[2]Department of Electronics and Information Systems (ELIS), Ghent University, Sint-Pietersnieuwstraat 41, 9000 Ghent, Belgium.

[3]Laboratoire d'Information Quantique (LIQ), Université libre de Bruxelles (U.L.B.), 50 Avenue F. D. Roosevelt, CP 225, B-1050 Bruxelles, Belgium

[*] Corresponding author: smassar@ulb.ac.be

The supplementary material is divided in two parts. In the first we provide a general introduction to the concept of Reservoir Computing, aimed towards readers who might want to realize an experimental implementation of this concept. In the second part we report some of the more technical details about the experiments described in the paper and the equipment used.

An Introduction to Reservoir Computing for Physicists and Engineers

Introduction

Understanding how physical systems can process information is a major scientific challenge. The tremendous progress that has been accomplished during the past century has given rise to whole new fields of science, such as computer science and digital (silicon) electronics, or quantum information. Information processing by classical analog systems is a case apart because remarkable examples are found in the biological sciences (e.g. the brain), but our understanding is still very incomplete. Thus, we understand at the basic level many aspects of how information is processed using biochemical reactions in cells, or how information is processed by neurons in a brain, to take two examples. But how these elementary processes are organized at a higher level, and what is the large scale architecture of these systems, still escapes us. Understanding these issues would be of great conceptual interest. It could also have huge technological repercussions, as it could completely change the way we build information processing machines. That this tremendous scope for progress exists is illustrated by the approximately 6 orders of magnitude gap in energy consumption between a brain and a present day silicon computer[1].

So far most work on information processing in analog systems has been based on imitating biological systems. This has given rise to the field of artificial neural networks. More abstract approaches have also been developed, such as hidden Markov models, or support vector machines. Reservoir computing [2, 3, 4, 5, 6, 7, 8] provides an alternative line of attack. The interest of reservoir computing is that 1) it proposes architectures that are quite different from those studied up to now; and 2) despite a relatively recent history -the first papers on the topic date from 2002-, its performances are comparable to, and sometimes even exceed, those of other approaches to machine learning[2]. A deeper understanding of reservoir computing could provide new insights into how analog classical systems, and in particular biological systems, process information. Additionally it could give rise to novel approaches for implementing computation and enable new applications.

The aim of the present text is not to duplicate existing reviews of reservoir computing [9, 10]. Rather, we wish to present the subject from the point of

[1] To simulate (with a speed reduction of 2 orders of magnitude) a cat brain that consumes roughly 1 Watt, the Blue Gene supercomputer consumes roughly a 10^6Watt. For a presentation of the simulation, see [1]

[2] Machine learning is the field of artificial intelligence concerned with developing algorithms whose response to the input -typically empirical data- improves with experience.

view of the physicist or engineer who wishes to build an experimental reservoir computer. We will argue that reservoir computing provides a quite detailed, but also quite flexible, road map towards building physical systems that process information with architectures very different from those used up to now.

Understanding Reservoir Computing

To understand the potentialities of reservoir computing, it is best to start with an example, and then examine to what extent the example can be generalized. Finally, on the basis of this discussion, we outline a road map for building experimental reservoir computers.

The archetypal reservoir computer is constructed as follows. It consists of a finite number N of internal variables $x_i(n)$ $(i = 0, ..., N-1)$ evolving in discrete time $n \in \mathbb{Z}$. The evolution of the variables $x_i(n)$ is perturbed by an external input $u(n)$. The evolution equations for the reservoir is:

$$x_i(n+1) = \tanh\left[\sum_{i=0}^{N-1} a_{ij} x_j(n) + b_i u(n)\right], \tag{1}$$

where the (time independent) coefficients a_{ij} and b_i are chosen independently at random from a simple distribution, for instance Gaussian distributions with mean zero and variances a^2 and b^2. The dynamics of this system (called the "reservoir") is fixed (i.e. the coefficients a_{ij} and b_i are fixed) . The dimensionality N of the reservoir is typically much larger than the input dimensionality. Given proper parameters a^2 and b^2, the reservoir will also have memory of past inputs. The use of the tanh nonlinearity in eq. (1) is traditional, but by no means essential, see below.

The aim of reservoir computing is to perform a computation on the input $u(n)$. To this end one maps the internal state of the reservoir to an output signal denoted $\hat{y}(n)$. The output should be as close as possible to a given target function $y(n)$. An example of this would be the detection of a stereotypical pattern in the input sequence such as spoken digits in a stream of audio, or a certain header in digital communication. Another example would be trying to predict future inputs such as electrical load prediction or financial time-series prediction.

The output $\hat{y}(n)$ of the reservoir computer at time n is a linear combination of the high dimensional internal states of the reservoir at time n:

$$\hat{y}(n) = \sum_{i=0}^{N-1} W_i x_i(n). \tag{2}$$

The W_i are usually chosen as follows. For some time interval, $n \in [1, T]$ we simulate the reservoir such that one knows both the inputs $u(n)$ and the target function $y(n)$. We minimize, over this training interval, the Mean Square Error between $\hat{y}(n)$ and $y(n)$:

$$\mathrm{MSE} = \frac{1}{T}\sum_{n=1}^{T}\left(\hat{y}(n) - y(n)\right)^2 . \tag{3}$$

This can be efficiently performed using standard linear regression techniques. [In cases where a classification of the inputs is desired, one may train several outputs (one for each class) and then use a winner takes all approach, or one may try to optimize the miss-classification rate using techniques such as logistic regression, Gaussian discriminant analysis, regression to the indicator function or a linear support vector machine.] Note that in general one should apply some form of regularization (to reduce the model complexity) to the readout function eq. (2). To this end one uses optimization techniques that prefer models with small parameters W_i. This is accomplished by using e.g. ridge regression.

The coefficients W_i are now fixed and the reservoir computer is ready for use: one can input into the reservoir new inputs $u(n)$ and obtain the corresponding output $\hat{y}(n)$. Because of the training procedure where we ensured proper generalization, the output will continue to be close to the target function, even for new data. It is considered good practice to check this. To this end one estimates the performance of the reservoir in a test phase during which new inputs are fed into the reservoir, and one compares, using the previously chosen W_i and an appropriate error metric, how close the output is to the target function.

Several remarks are now in order:

1. For good performance, it is necessary to scale the coefficients by a multiplicative factor

   $$a_{ij} \to \alpha a_{ij} \quad \text{and} \quad b_i \to \beta b_i \, .$$

   (In other words, if the coefficients are drawn from normal distributions with mean zero, one adjusts the variances of the normals). These parameters are often called the feedback gain and input gain respectively. The necessity for this scaling can be understood intuitively as follows: if the coefficients a_{ij} are too small, then the state equation (1) is strongly damped; if the coefficients a_{ij} are too large, then the system has highly complex dynamics or is even chaotic, and its behavior is very sensitive to small changes in the input. The coefficients b_i are adjusted in order to have an appropriate ratio between the contribution of the terms $\sum_j a_{ij} x_j(n)$ and the source term $b_i u(n)$. They also determine how non-linearly the inputs are expanded in the reservoir states.

2. Finding the optimal values of W_i is immediate in the case of linear regression. Indeed inserting eq. (2) into eq. (3) and extremizing with respect to W_i, one finds that the optimal choice is

   $$W_i = \sum_{j=0}^{N-1} R_{ij}^{-1} P_j \tag{4}$$

   where R_{ij}^{-1} is the inverse of the correlation matrix $R_{ij} = \frac{1}{T}\sum_{n=1}^{T} x_i(n) x_j(n)$ and $P_j = \frac{1}{T}\sum_{n=1}^{T} x_j(n) y(n)$. This should be contrasted to what would

happen if one tried to optimize the coefficients a_{ij} and b_i. In this case there are many more coefficients to optimize, and furthermore it is difficult to find the optimal value, as one may very easily end up in local optima. The use of a linear readout eq. (2) is thus highly advantageous.

3. Many variants on the above architecture have been investigated with success. We can list:

   (a) Using other nonlinearities than tanh, such as the biological inspired models which approximate the way real neurons interact with each other (spiking neuron models).

   (b) Using sparse connection matrices a_{ij} (i.e. matrices with most of the elements set to zero), so as to decrease the computational resources required to simulate the evolution.

   (c) Using multiple inputs $u_k(n)$, $k = 1, ..., K$. In this case one simply adds an index to the coefficients b_i. The source term in eq. (1) thus becomes $\sum_k b_{ik} u_k(n)$.

   (d) Using the output $\hat{y}(n)$ as input to the reservoir. This modification allows training the reservoir to behave as a given dynamical system [2, 5].

4. The above approach is extremely powerful. To take two examples:

   (a) Given a time series $u(n)$ for times $n = 1, ..., T$, the task is to guess the subsequent values of the time series $u(T+1), u(T+2), ...$ The time series could for instance come from the evolution of a chaotic dynamical system, such as the Mackey-Glass system. Using the trick 3d, results which outperform all other known approaches for predicting chaotic time series were obtained [5]. Reservoir computing also won an international competition on prediction of future evolution of financial time series [11].

   (b) Reservoir computing has been extensively applied to speech recognition. Using well known data sets, reservoir computing outperforms other approaches for isolated digit recognition [12, 13]. Recently reservoir computing has been applied to a more complex task: phoneme recognition. Performances comparable to the best alternative methods were obtained using 3 coupled reservoirs, each with $N = 20000$ variables [14].

Road map for building information processing systems

The basic setup of reservoir computing, although typically implemented in software, suggests many promising new avenues to implement computation in analog

dynamic systems. The theoretical requirements for reservoir computing to be computationally universal (in the analog sense) [4] are very loose: the reservoir is required to have fading memory, to be excitable by the input and a high dimensional readout must be possible. Many physical systems could be conceived that adhere to these rules and could thus potentially be turned in universal computing machines. However turning these general ideas into a working machine in more difficult.

If one wishes to build an experimental reservoir computer, then it is essential to understand what are the constraints, but also the design freedom. In this respect, a number of important lessons can be learned from the example presented in the previous section.

1. The fact that the interconnection matrix a_{ij} and the coefficients b_i in eq. (1) are chosen at random is extremely important. It means that fine tuning of a large number of coefficients is not necessary for a good reservoir computer. Rather almost all interconnection matrices will give good performance.

2. The fact that the tanh in eq. (1) can be changed into other non-linearities is also extremely important. It means that one does not have to imitate specific non-linear behavior (such as the specific dynamics of neurons in a biological brain), but one can use the nonlinearities that are easily accessible experimentally.

3. The fact that the only coefficients that are task specific are the W_i (the weights of the readout function) implies that a given dynamical system can be used for many different information processing tasks. It also implies that one can separate the design and analysis of the reservoir itself from the design and analysis of the readout function.

On the basis of these remarks, a reservoir computer can be built out of a dynamical system that satisfies the following constraints:

1. It should consist of a large number (say 50, or more, but this depends on the task and on the specifics of the dynamic system) dynamical variables which are coupled together by a non-linear evolution equation.

2. The evolution of the dynamical system can be perturbed by the external input.

3. As much as possible, one should try to break all symmetries of the system (this is the message coming from the fact that the a_{ij} and b_i are random: there is no residual structure/symmetry in the dynamics). To this end one should privilege dynamical systems that depend on a large number of random parameters.

4. A few global parameters must be experimentally tunable. These global parameters are used to adjust the operating point of the system (typically somwhat below the threshold of instability), and to adjust the overall

weight given to the external inputs (this corresponds to the scaling of the coefficients a_{ij} and b_i).

5. It should be possible to read-out the state of a large number (better all) of the dynamical variables, or at least to construct the readout function $y = \sum_i W_i x_i$ with adjustable weights W_i. Note that a reservoir computer can function satisfactorily even if only a subset of the dynamical variables are read out.

Once these constraints are satisfied, one can proceed to test the system (either using numerical simulations, or using the experimental realization). In the reservoir computing literature, there exists a series of (somewhat) standardized tests on which to evaluate the performance of the system. Some of these tests have a theoretical justification (e.g. linear memory capacity); others are interesting tasks which have been often used by the community. These tasks thus provide benchmarks with which to compare the performances of different reservoir computers. Possible tasks to study include:

1. Linear memory capacity and memory function [3]. In this task the target function $y(n) = u(n-k)$ is simply the input k time steps in the past. Performance on this task indicates how well the reservoir's state can be used to recover past inputs. If the linear memory capacity is small, then the reservoir will be unable to carry out tasks that require long memory of the input.

2. Non-linear memory capacity, see [15, 16]. In this task the target function is a non-linear function of the past inputs, for instance the product $y(n) = u(n-k)u(n-k')$. Performance on this task measures how much non-linear processing of the input is carried out by the reservoir.

3. Simulating specific Nonlinear Auto Regressive Moving Average (NARMA) systems. For the NARMA tasks, the aim is to use the reservoir to simulat the response of a NARMA system driven by a random input. Two variants are widely used: a 10th order system (NARMA10) introduced in [17], and the more difficult 30th order system (NARMA30) (see for instance [18] for a definition).

4. Predicting the evolution of the trajectory of chaotic attractors such as the attractor of the Mackey-Glass system [5].

5. Speech recognition: Several benchmarks have been published, going from Japanese vowel recognition [13], to isolated spoken digits [12], to phoneme recognition [14].

This (non-exhaustive) list of benchmarks provides a natural road-map for future experimental reservoirs. First master easy tasks such as signal classification, or isolated digit recognition. Then go on to moderately harder tasks such as NARMA10 or NARMA30. Finally, one can imagine tackling hard tasks such as phoneme recognition. In all cases, measure the linear and nonlinear memory

functions, as they will give task independent information on how the reservoir is processing information.

This road-map is appealing. Steps along it have been followed using reservoirs based on water waves in a bucket [19], on the cerebral cortex of a cat [20], on analog VLSI chips [21], using a (numerically simulated) array of coupled semi-conductor optical amplifiers [22]. However it is only very recently that it has been possible, using an electronics implementation and a delayed feedback loop, to demonstrate an analog reservoir computer with performance comparable to that of digital reservoirs on a non trivial task (in this case isolated digit recognition) [23].

Open questions and perspectives

The above road map leaves open a number of important questions. We outline the most obvious:

1. Effect of noise. In experimental realizations of reservoir computing, there will inevitably be noise and imperfections. How will these affect the performance of the scheme? There has not been an exhaustive analysis of this issue, but a few remarks are in order. One should distinguish between noise within the reservoir itself and noise in the readout.

   (a) Noise in the reservoir itself is deleterious. However, reservoir computers can continue to work even with moderate levels of noise. Indeed the noise can be viewed as an unwanted input, and the aim is to perform the desired task while ignoring this second unwanted input. A possible approach to counteract the effect of noise would simply be to increase the size of the reservoir. But this remains to be studied in detail.

   (b) Noise in the readout can be beneficial. Indeed adding noise to the readout is a trick, equivalent to ridge regularization, often used in numerical simulations to increase the robustness of reservoirs.

2. Best non-linearity. Experience with digital simulations of reservoirs shows that sigmoidal nonlinearities, such as tanh, give very good performance. However in experimental realizations, other nonlinearities may be much easier to implement. Experience suggests that the best non-linearity depends to some extent on the task at hand. Also sub-optimal nonlinearities can presumably be compensated by increasing the size of the reservoir.

3. Continuous time and continuous space. In numerical simulations, it is by far easier to work with a discrete set of variables $x_i(n)$, $i = 0, ..., N-1$ evolving in discrete time $n \in \mathbb{Z}$. But in experimental systems, it is often much more natural to work with continuous variables and continuous time. A theory of reservoir computing operating with continuous variables evolving in continuous time remains to be written, although some theoretical progress has been made along these lines [24].

Finally, what is the future of experimental reservoir computing? On the one hand the quest is fundamental: can we build analog systems that perform high-level information processing? Will we in this way gain new insights into how biological information processing systems operate? On the other hand experimental reservoir computers could ultimately have practical applications. Indeed they are based on architectures completely different from those used in present day digital electronics. Presumably the place to look for applications is at the frontier where digital electronics has difficulty coping, such as information processing at ultra high speeds, or with ultra low energy consumption, or for tasks which are hard to code using standard programming methods.

Experimental System, Numerical Simulations, Tasks

Here we describe in more detail our experimental system (schematized in Fig. 1 of the main text), the algorithm used to simulate it numerically, and the tasks we study to validate its performance as a reservoir.

Undriven system

We first consider the free running dynamics of our system, in the absence of input.

The light source consists of a NLK1556STB 20 mW laser diode at the wavelength 1564nm (in the standard C band of optical telecommunication). It produces a time independent intensity I_0.

The light passes through an intensity modulator (Photline MXAN-LN $LiNbO_3$ Mach-Zehnder modulator) driven by a time dependent voltage $V(t)$. The light intensity just after the modulator is $I(t) = I_0 \cos^2\left(\pi \frac{V(t)}{2V_\pi} + \phi\right)$ where V_π is a constant voltage (the voltage which is needed to go from a maximum to the next minimum of light at the output of the modulator, which in our case is $\approx 6V$) and ϕ can be adjusted by applying a DC bias voltage to the modulator. Taking $\phi = 3\pi/4 + \varphi$ allows us to rewrite

$$I(t) = \frac{I_0}{2} + \frac{I_0}{2} \sin\left(\frac{\pi V(t)}{V_\pi} + \varphi\right). \tag{5}$$

The light passes trough a tunable optical attenuator, enabling the adjustment of the loop's gain through the variation of the value I_0. It propagates through a long spool of fiber (1.7 km of single mode SMF-28 optical fiber in our experiment), and is then detected by a photodiode integrated with a transimpedance amplifier. The resulting voltage is amplified again (by a RF amplifier with a fixed gain of $27dB$) to produce the voltage $V(t)$ that drives the intensity modulator. In our experiment, the photodiode (a TTI TIA-525 with its transimpedance amplifier, nominal bandpass of 125 MHz) and RF amplifier operate in a linear regime, hence $V(t)$ is proportional to $I(t-T)$ where T is the round trip time of the oscillator. However, one must take into account that the photodiode acts as a lowpass filter and that the RF amplifiers act as highpass filters. Both the photodiode and the amplifiers also add noise. Thus, approximating the filters by first order filters, we have $\tilde{V}(\omega) = A \frac{\omega_0}{\omega_0 + j\omega} \frac{j\omega}{\omega_1 + j\omega} e^{j\omega T} \left(\tilde{I}(\omega) + \tilde{n}(\omega)\right)$ where j is the imaginary unit, tilde denotes the Fourier transform, A denotes the amplification factor, $\omega_1 \ll \omega_0$ are the cutoff frequencies of the resulting bandpass filter, and n(t) is the white noise. In the time domain we have $V(t) = A\mathcal{F}_{\omega_0\omega_1}(I(t-T) + n(t))$ where $\mathcal{F}_{\omega_0\omega_1}$ denotes the linear bandpass filter.

Because of the high pass filter, we can use as variable the fluctuations around the mean intensity. We define $x(t) = \frac{I(t)-I_0/2}{I_0/2}$ (i.e., we rescale the intensity I(t) to lie in the interval $[-1,+1]$) whereupon eq. (5) becomes

$$x(t) = \sin\left(\alpha \mathcal{F}_{\omega_0\omega_1}\left(x(t-T) + \frac{n(t)}{I_0}\right) + \varphi\right) \tag{6}$$

where $\alpha = \frac{\pi A I_0}{V_\pi}$ is experimentally adjustable by changing the input light intensity I_0. Experimentally it can be tuned over the range $\alpha \in [0, 4.2]$.

Typical values of the parameters are $T = 8.5\mu s$, $\omega_0/2\pi = 125$MHz, $\omega_1/2\pi = 50$kHz, and the amplitude of $n(t)/I_0$ is approximately 3.5%. At the output of the intensity modulator, an optical splitter enables us to take approximately 3% of the optical signal in order to measure the optical intensity in the fiber by the mean of a second amplified photodiode (another TIA-525). The resulting voltage is digitized with a National Instrument PXI card at the sample rate of 200 megasamples per second, or can be measured with a digital oscilloscope.

If we increase I_0 gradually, the intensity $I(t)$ undergoes a bifurcation diagram typical of nonlinear dynamical systems. Figure 1 shows the excellent agreement between the experimentally observed bifurcation diagram and the one obtained by numerical simulations of the evolution equations (with the bias $\varphi = 0$). In this bifurcation diagram, the number of bifurcations before reaching chaotic behavior is strongly affected by the amount of noise in the system. Comparing this bifurcation diagram to simulations is the most precise way we have to estimate the amount of noise in our experimental setup. The estimated value is in agreement with measurements carried out on each component of the setup separately. We also verified that the thickness of the branches inside the bifurcation diagram is mainly due to the noise level of oscilloscope.

Driven system

We now consider the addition of an input term. The input is characterized by a new time scale $T' \leq T$. The scalar input $u(n)$ (evolving in discrete time $n \in \mathbb{Z}$) and the input mask b_i are transformed into a continuous input $s(t)$ as follows:

$$s(t) = b_i u(n) \quad \text{for} \quad t \in \left[nT' + \frac{iT'}{N}, nT' + \frac{(i+1)T'}{N}\right], \quad i = 0, \ldots, N-1 \quad , \quad n \in \mathbb{Z} \tag{7}$$

where N is the number of nodes in the reservoir. The input mask values b_i, $i = 0, ..., N-1$ are randomly chosen from a given distribution (which may depend on the task). The time scale θ over which the input $s(t)$ changes is:

$$\theta = \frac{T'}{N}. \tag{8}$$

A voltage proportional to $s(t)$ is generated by a National Instruments PXI-5422 function generator (with a 16 bit resolution and a sample rate of 200

MSamples/s) and added to the output voltage of the amplified photodiode using a RF combiner placed at the entrance of the RF amplifier. So the voltage $V(t)$ that drives the intensity modulator is a combination of the light intensity and the input signal. The dynamical equations thus become

$$x(t) = \sin\left(\alpha \mathcal{F}_{\omega_0\omega_1}\left(x\left(t-T\right) + \frac{n\left(t\right)}{I_0}\right) + \beta \mathcal{F}_{\omega_1}\left(s\left(t\right)\right) + \varphi\right) \tag{9}$$

where β is experimentally adjustable by varying the output voltage amplitude of the function generator. In this equation, we take into account that the RF combiner is placed before the RF amplifier, and therefore that the source term is affected by the highpass filter $\mathcal{F}_{\omega_1}$ of the RF amplifier. Because of the difference between the time scales T and $\omega_1/2\pi$, the effect of the filter $\mathcal{F}_{\omega_1}$ is almost negligible. Its main effect is to ensure that the effective source signal $\mathcal{F}_{\omega_1}\left(s\left(t\right)\right)$ has mean value zero.

In our experiments we take $T = 8.5\mu s$. The number N of variables is taken in the range $50 - 200$. We take $T' = \frac{N}{N+1}T$ (see figure 2). Thus for $N = 50$, we have $\theta = 167ns$. The performance of the reservoir does not depend on the exact value of T' chosen, as long as T'/T is not a simple fraction (such as 1, 3/4, or 1/2), which would divide our reservoir into different independent subsystems.

Discretized dynamics

To obtain the discretized dynamics, we discretise the intensity along the fiber according to

$$x_i(n) \simeq x\left(nT' + \left(i + \frac{1}{2}\right)\theta\right) \quad i = 0, \ldots, N-1 = T'/\theta \tag{10}$$

where we suppose that $\theta = T'/N = T/\left(N+k\right)$ with k integer (see figure 2). We thus have that the physical time t is related to n, i, k through

$$t = nT' + \left(i + \frac{1}{2}\right)\theta = \frac{\left(nN + \left(i + \frac{1}{2}\right)\right)T}{N+k}. \tag{11}$$

Upon neglecting the effects of the filters $\mathcal{F}_{\omega_0}$ and $\mathcal{F}_{\omega_1}$ we obtain the discretized version of eq. (9), which reads

$$x_i\left(n\right) = \begin{cases} \sin\left(\alpha x_{i-k}\left(n-1\right) + \beta m_i u\left(n\right) + \varphi\right) & k \leq i < N \\ \sin\left(\alpha x_{N+i-k}\left(n-2\right) + \beta m_i u\left(n\right) + \varphi\right) & 0 \leq i < k \end{cases} \tag{12}$$

Note that the absence of synchronisation ($T' \neq T$, or equivalently $k > 0$) completely modifies the dynamics by coupling the discretised variables x_i to each other. Note also that the wrap around effect (the second line in eq. (12)) does not appear in traditional reservoir computing.

Numerical simulations

Two different numerical models were developed to study the capabilities of the network: a 'discretized' model, closer to the standard formulation of Echo State Networks, and a 'continuous' model, which is as close as possible to our experimental apparatus.

In the 'discretized' version of the model we implement the discretization described by eq. (12). No noise is considered, and the bandpass effects of the various components are neglected. The sine nonlinearity and the topology of the network are preserved. The optimal operating point of the system is found by tuning the parameters α and β in eq. (12). This model is used to set a performance goal for our experimental system: if the performance of the experiment is close to the one of the model, then our system is robust enough with respect to the noise and the effects introduced by each of its components. Moreover, the performance of the 'discretized' model is the same, within the experimental error, to the one of traditional networks as reported in [25], allowing us to validate the chosen nonlinearity and topology as good choices for a reservoir computer.

The 'continuous' Matlab model we developed is instead as close as possible to the experiment. All the signals in the simulation are discretized at 200 MSamples/s. This corresponds to the sample rate of the arbitrary waveform generator and the digitizer. All the components are represented by their transfer function at their respective operating point (sine function for the Mach-Zehnder modulator, responsivity and transimpedance gains of the photodiodes, gain of the amplifier). The collective frequency response of all the components is represented by a bandpass filter with first order slopes. This is a reasonable approximation to the exact slope of the bandpass filters which was measured using a vector network analyzer in open loop configuration. Noise is added to the signal at each noisy element of the system (dark current of the photodiode, noise added by each amplifier...).

The dynamics of the model correspond very closely to the experiment. This is illustrated for instance by the clear agreement of the simulated and measured bifurcation diagram (see fig. 1) and by the clear agreement of the simulated and measured performances on the channel equalization task, see Fig. 4 in the main text.

The continuous model allows us to easily explore the sensitivity to parameters, such as noise level or the shape of the nonlinearity, which can't always be reached with the experiment.

Post processing

The light intensity $I(t)$ in the fiber loop is converted to a voltage by a TIA-525 photodiode and recorded by the digitizer operating at 200MS/s. From the intensity $I(t)$ recorded during a time T' we extract N discrete variable values $x_i(n)$, $i = 0, ..., N-1$. This is carried out as follows. The intensity $I(t)$

is divided into N pieces of duration θ. We neglect the first quarter and the last quarter of the data points recorded over the duration θ and associate to $x_i(n)$ the average of the remaining data points. This procedure in which the beginning and end of each interval θ is omitted allows us to not be affected by the transients as the intensity goes from one value to the other, and also allows an efficient synchronization of our system. The estimator $\hat{y}(n)$ is then obtained by taking a linear combination $\hat{y}(n) = \sum_i W_i x_i(n)$ where the weights W_i are optimized. This post processing is carried out offline, on a computer. Fig. 2 shows an example of the input sent to the reservoir, the reservoir response and the discretization operated on the reservoir output.

Tasks

In our work we considered several tasks. We review them in detail.

Signal classification

This is a simple task that we use for a first evaluation of the performance of the reservoir. The input $u(n)$ to the system consists of random sequences of sine and square waves discretized into 12 points per period. The mask values b_i are drawn from the uniform distribution over the interval $[0, +1]$. The reservoir size is taken to be $N = 50$. The output $\hat{y}(n)$ should be 1 when the signal is the square wave, and 0 when it is the sine. The weights W_i are obtained by minimizing the MSE between $\hat{y}(n)$ and the ideal output. Experimentally we obtain NMSE $\simeq 1.5\ 10^{-3}$, which corresponds to essentially perfect operation for this task. These experimental results are in close agreement with those obtained using numerical simulations of our reservoir. This performance was obtained at an attenuation of -0.5 dB and an input gain of 10.5. For this task, we obtained the best results by shifting the bias $\varphi$to $\pi/4$.

For comparison, practically the same task was studied previously in [22]. An error rate (percentage of time the signal was misclassified) of 2.5% was obtained.

Nonlinear channel equalization

The task is to reconstruct the input $d(n) \in \{-3, -1, +1, +3\}$ of a noisy nonlinear wireless communication channel, given the output $u(n)$ of the channel. The relation between $d(n)$ and $u(n)$ is

$$\begin{aligned} q(n) \quad = \quad & 0.08d(n+2)\text{-}0.12d(n+1) + d(n) + 0.18d(n\text{-}1) \\ & \text{-}0.1d(n\text{-}2)) + 0.091d(n-3) – 0.05d(n-4) \\ & +0.04d(n\text{-}5) + 0.03d(n\text{-}6) + 0.01d(n-7) \end{aligned} \tag{13}$$

$$u(n) = q(n) + 0.036q(n)^2 - 0.011q(n)^3 + \nu(n) \tag{14}$$

where $\nu(n)$ represents i.i.d. Gaussian noise with zero mean, adjusted in power to yield signal-to-noise ratios ranging from 12 to 32 dB. This task was introduced in [26] and it was shown in [5] that reservoir computing could significantly improve performance on this task.

In our study, the input mask is taken to be uniformly distributed over the interval $[-1, +1]$. The reservoir size is taken to be $N = 50$. To perform the task, we first obtain an estimator $\hat{y}(n)$ of the input by minimizing the MSE between $\hat{y}(n)$ and $d(n)$. We then obtain an estimator $\hat{d}(n)$ by replacing $\hat{y}(n)$ by the discretized value $\{-3, -1, +1, +3\}$ to which it is closest. Finally we estimate the Symbol Error Rate (SER), i.e., the fraction of time that $\hat{d}(n)$ differs from $d(n)$. The performance of the network is calculated as the average performance over 10 different input sequences, in which the first 3000 samples form the training set and the following 6000 samples form the test set. The SER on the test set is then studied as a function of Signal to Noise Ratio at the input. The values reported in Figure 4 in the main text are the average SER for 10 different trials, while the error bars represent the standard deviation of the SER for the same trials. It should be noted that 6000 test steps are the maximum number of steps that the arbitrary waveform generator in our setup allows. Hence, when SERs approach 10^{-4}, our average SERs include trials where two errors, one error, or no error at all have been made by the reservoir. This means that the error bars on the experimental data where SERs are close to 10^{-4} are only roughly estimated. In contrast, data from simulations do not suffer from this effect, as we can arbitrarily increase the number of test samples for a more precise measurement.

For comparison, at a SNR ratio of 28dB, the three models studied in [26] gave SER of $2 \cdot 10^{-3}$,$4 \cdot 10^{-3}$,$1.5 \cdot 10^{-2}$, while the reservoir studied in [5] gave SERs of $1 \cdot 10^{-4}$ to $1 \cdot 10^{-5}$. At the same SNR of 28dB, our experimental system gives a SER of $1.3 \cdot 10^{-4}$.

The optimal working point for this test actually depends on the signal-to-noise ratio. For SNRs of 32, 28 and 24 dB the best working point was at 1.66 input gain, -4.2 dB attenuation. For a SNR of 20 dB the best working point was at 1.23 input gain, -3.5 dB attenuation; for a SNR of 16 dB it was at 0.85 input gain and -2.8 dB attenuation; finally, for a SNR of 12 dB the best working point was at 1.44 input gain, -2.8 dB attenuation.

NARMA10

In this task the aim is to reproduce the behavior of a nonlinear, tenth-order system with random input drawn from a uniform distribution over the interval $[0, 0.5]$. The equation defining the target system is given by

$$y(n+1) = 0.3y(n) + 0.05y(n)\left(\sum_{i=0}^{9} y(n-i)\right) + 1.5u(n-9)u(n) + 0.1 \quad (15)$$

For this task, the mask is uniformly distributed over $[0, +1]$ and the reservoir

size is $N = 50$. We first train our system using 1000 time steps, then we test the system on a new sequence of 1000 inputs. The performance of the reservoir is measured by the Normalized Mean Square Error of the estimator $\hat{y}(n)$, averaged over 10 different pairs of train and test sequences. For this NARMA10 task, the best performances (NMSE = 0.16 for the discretized simulation, 0.168 for the continuous simulation and 0.167 for the experiment) are obtained in a very linear regime, at an input gain of 0.55 and an attenuation of -0.5 dB.

Isolated spoken digit recognition

For the isolated spoken digit recognition task, the data is taken from the NIST TI-46 corpus [27]. It consists of ten spoken digits (0...9), each one recorded ten times by five different female speakers. These 500 spoken words are sampled at 12.5 kHz. This spoken digit recording is preprocessed using the Lyon cochlear ear model [28]. The input to the reservoir $u_j(n)$ consists of an 86-dimensional state vector ($j = 1, ..., 86$) with up to 130 time steps. The number of variables is taken to be $N = 200$. The input mask is taken to be a $N \times 86$ dimensional matrix b_{ij} with elements taken from the the set $\{-0.1, +0.1\}$ with equal probabilities. The product $\sum_j b_{ij} u_j(n)$ of the mask with the input is used to drive the reservoir. Ten linear classifiers $\hat{y}_k(n)$ ($k = 0, ..., 9$) are trained, each one associated to one digit. The target function for $y_k(n)$ is $+1$ if the spoken digit is k, and -1 otherwise. The classifiers are averaged in time, and a winner-takes-all approach is applied to select the actual digit.

In our study, the 500 spoken words are divided in five subsets. We trained the reservoir on four of the subsets, and then tested it on the fifth one. This is repeated five times in order to use each subset as test part. In this way we can test our system over all the speakers and digits, and compute an average performance . The performance is given in terms of the Word Error Rate, that is the fraction of digits that are misclassified. We obtain a WER of 0.4% which correspond to 2 errors in 500 recognized digits; the best operating point is at an input gain of 1.66 and a feedback gain of -0.9 dB .

For comparison, in [29], where Reservoir Computing was first used on this spoken digit benchmark, a WER of 4.3% was reported for a reservoir of size 1232. In [12] a WER of 0.2% was obtained for a reservoir of size 308 and using the winner-takes-all approach. In [30] a WER of 1.3% for reservoirs of size 200 is reported. The experimental reservoir reported in [23] gives a WER of 0.2% using a reservoir of size 400. The Sphinx-4 system [31] uses a completely different method based on Hidden Markov Models and achieves a WER of 0.55% on the same data set.

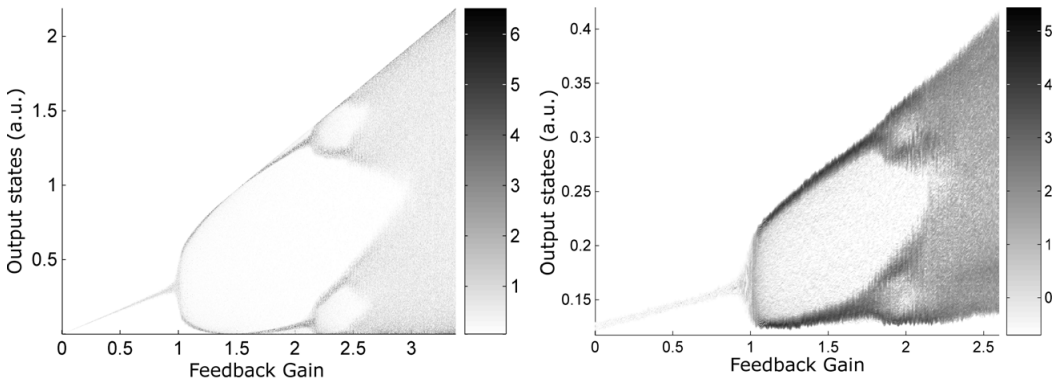

Figure 1: Simulated (left panel) and measured (right panel) bifurcation diagram. The vertical axis (output states) is proportional to the optical intensity in the optical fiber for the simulations, and to the voltage at the output of the readout photodiode for the measurements (this voltage is equal to de optical intensity multiplied by the gain of the amplified photodiode). The gray scale represents the histogram of optical intensities inside the system as a function of the feedback gain. When the feedback gain is lower than unity, only one value of the light intensity is possible. For feedback gain slightly larger than unity, the light intensity oscillates between two values. For even larger feedback gain (around 2), the nonbijective nature of the Mach-Zehnder modulator's transfer function leads to oscillation between multiple light intensity levels or even to a chaotic behavior. The number of bifurcations before reaching the chaotic behavior is determined by the amount of noise inside the system. The thickness of the branches in the measured bifurcation diagram is due to the noise added by the oscilloscope.

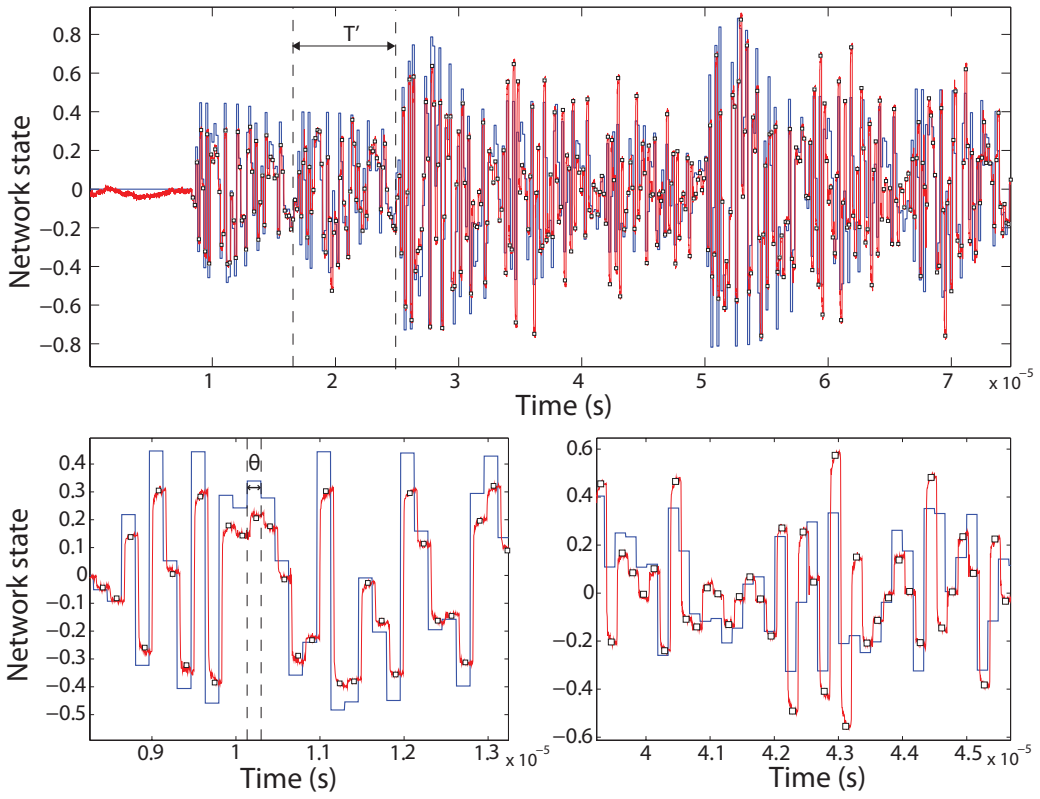

Figure 2: Operation of the experimental reservoir. The blue line represents the masked input driving the reservoir; the red line represents the reservoir response to the depicted input as measured by the oscilloscope. Both have been normalized so that their values lie between -1 and 1 for the whole length of the experiment. The white squares along the red line represent the discretized values of the reservoir states, obtained by averaging the reservoir output on a time interval of duration$\theta/2$. The upper part of the figure shows the reservoir operation for the first 8 inputs of the reservoir; note that the input is zero up to $t \approx 0.8 \cdot 10^{-5}$. The bottom left panel shows a detail from the measurement on the response to the very first input value, when no feedback is present yet, and we only see the instantaneous response of the Mach-Zehnder. The panel on the bottom right depicts the response of the system to the 5th input, where the effect of the feedback is clearly visible: the amplitudes of the reservoir states are no longer just related to the instantaneous input, but are influenced by previous inputs as well.

REFERENCES

References

[1] Ananthanarayanan, R., Esser, S. K., Simon, H. D., and Modha, D. S. The cat is out of the bag: cortical simulations with 1^09 neurons, 10^13 synapses. *Proc. Int. Conference for High Performance Computing, Networking, Storage and Analysis (SC '07)*, 1–12, (2007).

[2] Jaeger, H. The "echo state" approach to analysing and training recurrent neural networks. *Technical Report GMD Report* **148**, German National Research Center for Information Technology (2001).

[3] Jaeger, H. Short term memory in echo state networks. *Technical Report GMD Report* **152**, German National Research Center for Information Technology (2001).

[4] Maass, W., Natschlager, T., and Markram, H. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Comput.*, **14**, 2531–2560 (2002).

[5] Jaeger, H. and Haas, H. Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication. *Science (New York, N.Y.)*, **304**, 78–80 (2004).

[6] Steil, J. J. Backpropagation-decorrelation: online recurrent learning with O(N) complexity. *Proceedings of IJCNN '04*, **1**, 843–848 (2004)

[7] Legenstein, R. and Maass., W. What makes a dynamical system computationally powerful?, In Haykin, S., Principe, J., Sejnowski, T., McWhirter, J., editors, *New Directions in Statistical Signal Processing: From Systems to Brain*, pages 127–154. MIT Press (2005).

[8] Verstraeten, D., Schrauwen, B., D'Haene, M., and Stroobandt, D. An experimental unification of reservoir computing methods. *Neural Netw.*, **20**, 391–403 (2007).

[9] Lukoševičius, M. and Jaeger, H. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, **3**, 127–149 (2009).

[10] Hammer, B., Schrauwen, B., and Steil, J. J. Recent advances in efficient learning of recurrent networks. In *Proceedings of the European Symposium on Artificial Neural Networks*, 213–216 (2009).

[11] http://www.neural-forecasting-competition.com/NN3/index.htm.

[12] Verstraeten, D., Schrauwen, B., and Stroobandt, D. Reservoir-based techniques for speech recognition. In *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, 1050–1053 (2006).

[13] Jaeger, H., Lukosevicius, M., Popovici, D. and Siewert, U. Optimization and applications of echo state networks with leaky-integrator neurons. *Neural Netw.*, **20** , 335–52 (2007).

[14] Triefenbach, F., Jalalvand, A., Schrauwen, B., and Martens, J. Phoneme recognition with large hierarchical reservoirs. *Advances in Neural Information Processing Systems*, **23**, 1–9 (2010).

[15] Verstraeten, D., Dambre, J., Schrauwen, B., and Massar, S. Linear and nonlinear memory capacity of dynamical systems. *in preparation*.

[16] Verstraeten, D., Dambre, J., Dutoit, X., and Schrauwen, B. Memory versus non-linearity in reservoirs. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, 1–8 (2010).

[17] Atiya, A. F. and Parlos, A. G. New results on recurrent network training: unifying the algorithms and accelerating convergence. *IEEE T. Neural Netw.*, **11**, 697–709 (2000).

[18] Schrauwen, B., Wardermann, M., Verstraeten, D., Steil, J. J., and Stroobandt, D. Improving reservoirs using intrinsic plasticity. *Neurocomputing*, **71**, 1159–1171 (2008).

[19] Fernando, C. and Sojakka, S. Pattern recognition in a bucket. In Banzhaf, W., Ziegler, J., Christaller, T., Dittrich, P. and Kim, J., editors, *Advances in Artificial Life*, volume 2801 of *Lecture Notes in Computer Science*, 588–597. Springer Berlin / Heidelberg (2003).

[20] Nikolic, D., Häusler, S., Singer, W., and Maass, W. Temporal dynamics of information content carried by neurons in the primary visual cortex. *Advances in Neural Information Processing Systems*, **19**, 1041–1048 (2007).

[21] Schürmann, F., Meier, K., and Schemmel, J. Edge of chaos computation in mixed-mode vlsi - a hard liquid. In *Advances in Neural Information Processing Systems*. MIT Press (2005).

[22] Vandoorne, K. *et al.* Toward optical signal processing using photonic reservoir computing. *Opt. Express*, **16**, 11182–92 (2008).

[23] Appeltant, L. *et al.* Information processing using a single dynamical node as complex system. *Nat. Commun.*, **2**, 468 (2011).

[24] Hermans, M., and Schrauwen, B. Memory in linear recurrent neural networks in continuous time. *Neural networks : the official journal of the International Neural Network Society*, **23**, 341–55 (2010).

[25] Rodan, A. and Tino, P. Simple deterministically constructed recurrent neural networks. *Intelligent Data Engineering and Automated Learning - IDEAL 2010*, 267–274 (2010).

[26] Mathews, V. J. and Lee, J. Adaptive algorithms for bilinear filtering. *Proceedings of SPIE*, **2296**, 317–327 (1994).

[27] Texas Instruments-Developed 46-Word Speaker-Dependent Isolated Word Corpus (TI46), September 1991, NIST Speech Disc 7-1.1 (1 disc) (1991).

[28] Lyon, R. A computational model of filtering, detection, and compression in the cochlea. In *ICASSP '82. IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 1282–1285. IEEE (1982).

[29] Verstraeten, D., Schrauwen, B., and Stroobandt, D. Isolated word recognition using a liquid state machine. In *Proceedings of the 13th European Symposium on Artificial Neural Networks (ESANN)*, 435–440 (2005).

[30] Rodan, A. and Tino, P. Minimum complexity echo state network. *IEEE T. Neural Netw.*, **22**, 131–44 (2011).

[31] Walker, W. *et al.* Sphinx-4: a flexible open source framework for speech recognition. Technical report, Mountain View, CA, USA (2004).