Towards pattern generation and chaotic series prediction with photonic reservoir computers

Piotr Antonik[a], Michiel Hermans[a], François Duport[b], Marc Haelterman[b], and Serge Massar[a]

[a]Laboratoire d'Information Quantique, Université Libre de Bruxelles, 50 Avenue F. D. Roosevelt, CP 225, 1050 Brussels, Belgium
[b]Service OPERA-Photonique, Université Libre de Bruxelles, 50 Avenue F. D. Roosevelt, CP 194/5, 1050 Brussels, Belgium

ABSTRACT

Reservoir Computing is a bio-inspired computing paradigm for processing time dependent signals that is particularly well suited for analog implementations. Our team has demonstrated several photonic reservoir computers with performance comparable to digital algorithms on a series of benchmark tasks such as channel equalisation and speech recognition. Recently, we showed that our opto-electronic reservoir computer could be trained online with a simple gradient descent algorithm programmed on an FPGA chip. This setup makes it in principle possible to feed the output signal back into the reservoir, and thus highly enrich the dynamics of the system. This will allow to tackle complex prediction tasks in hardware, such as pattern generation and chaotic and financial series prediction, which have so far only been studied in digital implementations. Here we report simulation results of our opto-electronic setup with an FPGA chip and output feedback applied to pattern generation and Mackey-Glass chaotic series prediction. The simulations take into account the major aspects of our experimental setup. We find that pattern generation can be easily implemented on the current setup with very good results. The Mackey-Glass series prediction task is more complex and requires a large reservoir and more elaborate training algorithm. With these adjustments promising result are obtained, and we now know what improvements are needed to match previously reported numerical results. These simulation results will serve as basis of comparison for experiments we will carry out in the coming months.

Keywords: Reservoir computing, FPGA, opto-electronic systems, pattern generation, chaotic series prediction

1. INTRODUCTION

Reservoir Computing (RC) is a set of methods for designing and training artificial recurrent neural networks.[1,2] A typical reservoir is a randomly connected fixed network, with random coupling coefficients between the input signal and the nodes. This reduces the training process to simply solving a system of linear equations.[3,4] The RC algorithm has been successfully applied to phoneme recognition,[5] for instance, and won an international competition on prediction of future evolution of financial time series.[6]

The simplicity of Reservoir Computing makes it very well suited for analog implementations: various electronic,[7,8] opto-electronic[9–11] and all-optical[12–16] implementations have been reported since 2012. These realisations have been successfully applied to channel equalisation and speech recognition tasks, among others, with performances comparable to other digital algorithms. We have recently reported the first online-trained opto-electronic reservoir computer.[17] The key feature of this implementation is the FPGA chip, programmed to generate the input sequence, train the reservoir computer using the simple gradient descent algorithm, and compute the reservoir output signal in real time.

This setup also offers the possibility to feed the output signal back into the reservoir by programming the FPGA chip to sum up the input and output signals. In this work, we investigate numerically the performance of the system[17] with output feedback. Feeding the output signal back into the reservoir allows to tackle prediction tasks, which are impossible to solve without such feedback. We test our reservoir computer on two examples of

Further author information: (Send correspondence to P.A.)
P.A.: E-mail: pantonik@ulb.ac.be

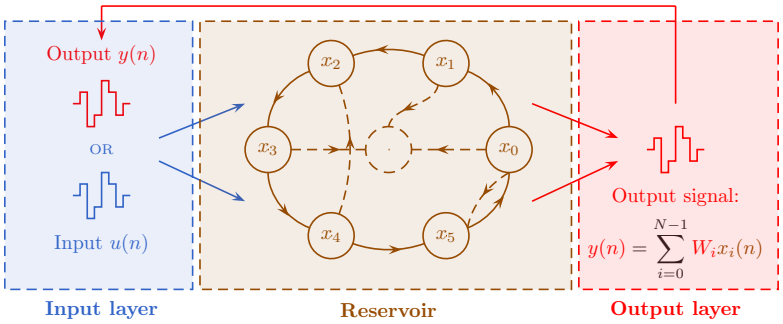


Figure 1. Schematic representation of a reservoir computer with output feedback. Brown lines represent a general recurrent neural network with random interconnections, solid lines highlight a network with ring topology, used here. The time multiplexed input signal $u(n)$ is injected into a reservoir with N nodes (in this scheme $N = 6$), denoted $x_i(n)$. The output signal $y(n)$, given by a linear combination of the readout weights w_i with the reservoir states $x_i(n)$, is fed back into the network.

prediction tasks: pattern generation[18] and chaotic series prediction.[19] The first can be directly applied to motion generation and robot control, or indirectly to data storage.[20] For the second task, we chose the Mackey-Glass chaotic system, which has already been investigated numerically.[1] The goal of these numerical experiments was twofold: evaluate the performance of the existing experiment on these two tasks, and find out what enhancements could be made to improve the results. We show that pattern generation requires little modifications and good results can be achieved with the current system. The Mackey-Glass prediction task, on the other hand, requires more complex training algorithms and much larger reservoirs to be solved efficiently. We investigate three training methods, different reservoir sizes, and obtain performances comparable to previous numerical studies.[1] The results presented here will be used as a starting point for future experimental implementations.

2. BASIC PRINCIPLES

2.1 Reservoir Computing

A general reservoir computer is a recurrent neural network with random fixed internal and input connections.[3] In our implementation, depicted in figure 1, we use a sine function $f = \sin(x)$ as nonlinearity,[9,10] and a ring topology[7,21] to simplify the interconnection matrix of the network, so that only the first neighbour nodes are connected. Under these conditions the evolution equations of the network are given by

$$x_0(n+1) = \sin\left(\alpha x_N(n-1) + \beta M_0 u(n)\right), \tag{1a}$$
$$x_i(n+1) = \sin\left(\alpha x_{i-1}(n) + \beta M_i u(n)\right), \tag{1b}$$

where $x_i(n)$, $i = 0, \ldots, N-1$ are the internal variables, evolving in discrete time $n \in \mathbb{Z}$, α and β parameters are used to adjust the feedback and the input signals, respectively, $u(n)$ is a time multiplexed input signal, and M_i is the input mask, drawn from a uniform distribution over the interval $[-1, +1]$.[9,12,21] The reservoir computer produces an output signal $y(n)$ given by a linear combination of the states of its internal variables

$$y(n) = \sum_{i=0}^{N} w_i x_i(n), \tag{2}$$

where $x_N = 1$ is a constant neuron used to adjust the bias of the output signal, w_i are the readout weights, trained either offline (see section 2.2.1), or online (as described in sections 2.2.2 and 2.2.3) in order to minimise the mean square error between the output signal $y(n)$ and the target signal $d(n)$. When the output signal is fed back into the reservoir (see section 2.3), the dynamics of the systems is described by the following equations

$$x_0(n+1) = \sin\left(\alpha x_N(n-1) + \beta M_0 y(n)\right), \tag{3a}$$
$$x_i(n+1) = \sin\left(\alpha x_{i-1}(n) + \beta M_i y(n)\right). \tag{3b}$$

2.2 Training Methods

A reservoir computer can be trained using various methods. In this work, we used two online training algorithms, simple gradient descent and recursive least squares, as well as the common offline method for the sake of comparison.

2.2.1 Offline training

The optimal readout weights w_i can be computed offline, that is, after running the experiment and recovering all the reservoir states $x_i(n)$. To minimise the mean square error (MSE), given by

$$\mathrm{MSE} = \frac{1}{T}\sum_{n=1}^{T}\left(d(n) - y(n)\right)^2, \tag{4}$$

one needs to insert equation (2) into (4) and differentiate with respect to w_i. This gives a system of N linear equations admitting the following solution

$$w_i = \sum_{j=0}^{N-1} R_{ij}^{-1} P_j, \tag{5}$$

where R_{ij}^{-1} is the inverse of the correlation matrix

$$R_{ij} = \frac{1}{T}\sum_{n=1}^{T} x_i(n) x_j(n), \tag{6}$$

and P_j is the cross-correlation vector

$$P_j = \frac{1}{T}\sum_{n=1}^{T} x_j(n) d(n). \tag{7}$$

This method has been used in most experimental implementation so far.[7,9,10,12,13,15,16]

2.2.2 Simple gradient descent

The gradient, or steepest, descent method is an algorithm for finding a local minimum of a function using its gradient.[22] The rule for updating the readout weights w_i is given by[23]

$$w_i(n+1) = w_i(n) + \lambda\left(d(n) - y(n)\right) x_i(n), \tag{8}$$

where λ is a constant learning rate.[17]

2.2.3 Recursive least squares

The Recursive Least Squares (RLS) algorithm[24] can be used for iterative computation of the inverse of the correlation matrix R_{ij}^{-1} and the readout weights w_i. It can be summarised as follows

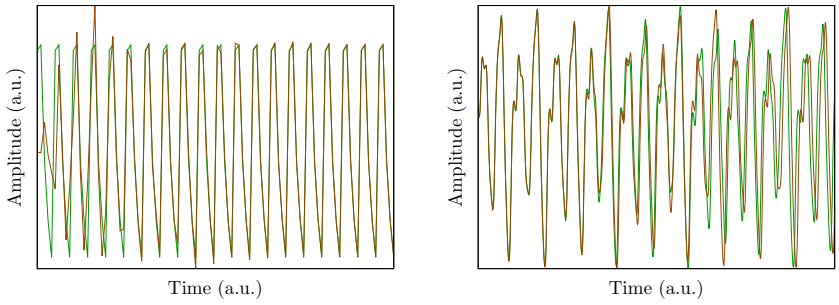
$$k(n) = \frac{\nu^{-1}\Gamma(n-1)x(n)}{1 + \nu^{-1}x^T\Gamma(n-1)x(n)}, \tag{9a}$$

$$w(n) = w(n-1) + k(n)\left(d(n) - y(n)\right), \tag{9b}$$

$$\Gamma(n) = \nu^{-1}\Gamma(n-1) - \nu^{-1}k(n)x^T(n)\Gamma(n-1), \tag{9c}$$

where $k(n)$ is called the gain vector and $\Gamma(n)$ is the estimate of the inverse of the correlation matrix. The forgetting factor ν is set to 1.

This method ensures faster convergence of the readout weights but is much more computationally intensive than the simple gradient descent algorithm. It is also much harder to implement on the FPGA chip, as it requires one to program vector multiplication and scalar division using bit logic.



(a) (b)

Figure 2. Illustrations of the two tasks considered here. **(a)** Training process of the reservoir computer for the pattern generation task. The output signal (brown line) randomly oscillates at the beginning, and matches the desired pattern (green curve) by the end. **(b)** Reservoir output (brown curve) and target (green line) signals during the test phase of the Mackey-Glass prediction task. While the output signal matches the chaotic series in the beginning, it slowly drifts away during the experiment.

2.3 Tasks

We tested our system on two examples of prediction tasks : pattern generation and chaotic series prediction. For the second task, we chose to work with a chaotic system described by Mackey-Glass equations.[25]

2.3.1 Pattern generation

A pattern is a short sequence of randomly chosen real numbers (here within the interval $[-0.5, 0.5]$) that is repeated periodically to form an infinite time series. As will be shown in section 4.1, the length of the sequence may be set from 1 up to the number of neurons N. During the training phase, the reservoir computer receives the pattern signal as input and is trained to predict the next value of the pattern from the current one. The length of the training input sequence is measured in terms of the pattern length, that is, how many times the whole pattern is presented to the reservoir computer. The duration of the training process depends on the length of the pattern. While short patterns are successfully learnt after 100 repetitions, long patterns need to be shown up to $50k$ times. During training, no output signal is fed back into the reservoir.

After the training phase, the reservoir input is switched from the training sequence to the reservoir output signal, and the system is left running autonomously. The aim is to obtain a stable pattern generator, that reproduces precisely the training sequence and doesn't deviate to another periodic behaviour. To measure the stability of the generator, we compute the MSE between the reservoir output signal and the target pattern signal at the end of both training and test phases. If the error doesn't grow during the test phase, the system is considered to exhibit stable behaviour.

Figure 2(a) depicts the training process for this task. The desired pattern is shown in green, the brown line corresponds to the reservoir output. The systems exhibits random oscillations at the beginning of the training (left-hand side of the plot) and quickly learns the pattern, so that the two lines match on the right-hand side of the figure.

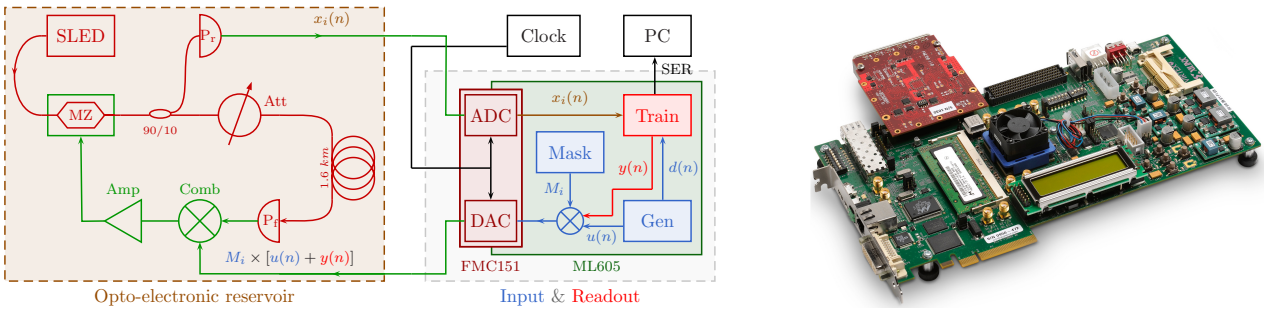
2.3.2 Mackey-Glass chaotic series prediction

The Mackey-Glass delay differential equation

$$\frac{dx}{dt} = \beta \frac{x(t-\tau)}{1+x^n(t-\tau)} - \gamma x \tag{10}$$

with τ, γ, β, $n > 0$ was introduced to illustrate the appearance of complex dynamics in physiological control systems.[25] To obtain chaotic dynamics, we set the parameters as follows:[1] $\beta = 0.2$, $\gamma = 0.1$, $\tau = 17$ and $n = 10$.

Training and test phases are the same as for pattern generation. During training, the reservoir receives the time series and is trained to perform a one-step ahead prediction. During the test phase, the reservoir receives



(a) (b)

Figure 3. **(a)** Schematic representation of the simulated setup, based on the experimental system.[17] Optical and electronic components of the opto-electronic reservoir are shown in red and green, respectively. It contains an incoherent light source (SLED), a Mach-Zehnder intensity modulator (MZ), a 90/10 beam splitter, an optical attenuator (Att), an approximately 1.6 km fibre spool, two photodiodes (P_r and P_f), a resistive combiner (Comb) and an amplifier (Amp). The FPGA board implements both the input and output layers, generating the input symbols and training the readout weights. The reservoir output signal $y(n)$ is added to the input signal $u(n)$, then multiplied by the input mask M_i and sent to the opto-electronic reservoir. The computer controls the devices and records the results. **(b)** Xilinx ML605 board with Virtex 6 FPGA chip and 4DSP FMC150 daughter card (FMC150 and FMC151 cards look practically the same).

its own output and the system runs autonomously. The MSE is used to evaluate the training phase. We also compute the number of correct prediction steps made by the reservoir, that is, the number of error values below a certain MSE threshold, to measure the quality of emulation of the Mackey-Glass system.

Figure 2(b) shows the behaviour of the reservoir computer on the Mackey-Glass prediction task during the test phase, right after the end of the training. The output signal (brown curve) matches the Mackey-Glass series (green curve) at the beginning, but slowly deviates from the target with time.

3. EXPERIMENT AND SIMULATIONS

3.1 Experimental Setup

Figure 3(a) depicts the experimental setup,[17] which is the basis for numerical simulations presented here. It consists of two main components: the opto-electronic reservoir and the FPGA board.

3.1.1 Opto-electronic reservoir

The opto-electronic reservoir is based on previously published works.[9,10,17] The reservoir states are encoded into the intensity of incoherent light signal, produced by a superluminiscent diode (Thorlabs SLD1550P-A40). The Mach-Zehnder (MZ) intensity modulator (Photline MXAN-LN-10) implements the nonlinear function, its operating point is adjusted by applying a bias voltage, produced by a Hameg HMP4040 power supply. A fraction (10%) of the signal is extracted from the loop and sent to the readout photodiode (TTI TIA-525I) and the resulting voltage signal is sent to the FPGA. An optical attenuator (JDS HA9) is used to set the feedback gain α of the system (see equations (1) and (3)). The fibre spool consists of approximately 1.6 km single mode fibre, giving a round trip time of 7.94 µs. The resistive combiner sums the electrical feedback signal, produced by the feedback photodiode (TTI TIA-525I), with the input signal from the FPGA to drive the MZ modulator, with an additional amplification stage of +27 dB (ZHL-32A+ coaxial amplifier) to span the entire V_π interval of the modulator.

A personal computer is used to control the hardware and setup the parameters between experiments. It executes a Matlab script, designed to run the experiment multiple times over a set of predefined values of parameters of interest and select the combination that yields the best results.

3.1.2 FPGA board

The reservoir computer is trained by a Xilinx ML605 evaluation board (see figure 3(b)), powered by a Virtex 6 XC6VLX240T FPGA chip. The board is paired with a 4DSP FMC151 daughter card, containing one two-channel ADC (Analog-to-Digital converter) and one two-channel DAC (Digital-to-Analog converter). The ADC's maximum sampling frequency is 250 MHz with 14-bit resolution, while the DAC can sample at up to 800 MHz with 16-bit precision.

The FPGA is programmed to execute the simple gradient descent algorithm (see section 2.2.2). It receives the neurons $x_i(n)$, sampled by the ADC, and computes the reservoir output signal $y(n)$. The target signal $d(n)$ is then used to compute the error and adjusts the readout weights w_i following equation (8). The chip can also be programmed to perform the RLS algorithm, described in section 2.2.3.

The arithmetic operations mentioned above are performed on real numbers. However, a FPGA is a logic device, designed to operate bits. The performance of the design thus highly depends on the bit-representation of real numbers, i.e. the precision. The main limitation comes from the DSP48E slices, used to perform multiplications. These blocks are designed to multiply a 25-bit integer by a 18-bit integer. To meet these requirements, our design uses a fixed-point representation with different bit array lengths for different variables. Parameters and signals that stay within the $]-1,1[$ interval are represented by 18-bit vectors, with 1 bit for the sign and 17 for the decimal part. These are the learning algorithm parameters, the input mask elements M_i and the reservoir states $x_i(n)$, extended from the 14-bit ADC output. Other variables, such as reservoir output $y(n)$ and readout weights w_i span a wider $[-16,16]$ interval and are represented as 25-bit vectors, with 1 sign bit, 4 bits for the integer part and 20 bits for the decimal part.

In order to train the reservoir computer offline (using method described in section 2.2.1), the FPGA can be programmed to record the reservoir states $x_i(n)$ and send them to the personal computer through an Ethernet connection. The readout weights w_i can then be uploaded on the chip for real-time computation of the reservoir output signal $y(n)$ during the test phase. Such setup makes less use of the FPGA chip, but offers more control and precision over the training process, which will make it a good starting point for future experimental studies.

3.2 Numerical Simulations

All numerical experiments were performed in Matlab. Most simulations were computed on a standard personal computer, some computationally-intensive scripts with long training sets or large reservoirs were executed on a dedicated computer with a 12-core CPU and 64 Gb RAM. The simulations account for major aspects of our experimental setup and allow to scan the most influential parameters, such as input gain β, feedback gain α, learning rate λ, reservoir size N, training length or precision. In the next sections we overview several considerations specific for each task.

3.2.1 Pattern generation

For this task, we sough for optimal values of the input and feedback gain parameters β and α, we checked the minimal FPGA precision required for sufficiently low training MSE, and also investigated the maximum pattern length L that could be generated with a 50-neuron reservoir.

The reservoir computer was trained to perform a two-step ahead prediction of the pattern, to account for the computational delay of the reservoir output signal. Indeed, sampling the reservoir states $x_i(n)$ and computing the reservoir output signal $y(n)$ is not instantaneous. That is, when input value $M_1 \times u(n)$ is sent to the first neuron $x_1(n)$ in the opto-electronic reservoir, the FPGA has just finished sampling the previous reservoir state $x_{50}(n-1)$, and has yet to compute the output value $y(n-1)$. For this reason, the reservoir output feedback is delayed by one roundtrip time T.

3.2.2 Mackey-Glass series prediction

The Mackey-Glass equation (10) was integrated using the RK4 method,[26] with step $h = 0.1$. The resulting time series was shifted by -1 and passed through a tanh function in order to fall within the $[0.5, 0.3]$ interval.[1] For faster simulations, an input sequence of 3×10^6 values was computed and saved in a file.

For this task, we also investigated the optimal values of gain parameters α and β, as well as the size of the reservoir N, the choice of the training algorithm and its precision. For prediction length estimation, we set the threshold error at $\mathrm{MSE}_{\mathrm{th}} = 10^{-3}$.

The issue with the delayed output signal $y(n)$, discussed in the previous section, was solved in a different way here. In fact, performing a two-step ahead prediction on the Mackey-Glass chaotic series results in a very poor performance. However, given the randomness of the input mask M_i, it is possible to set first value to $M_0 = 0$ without noticeable loss of performance. This trick allows to delay the injection of the output signal back into the reservoir by one neuron duration. For a 250-neuron reservoir in a 8 µs delay loop (this is the maximum reservoir size achievable with this delay loop and our ADC) this offers a delay of 32 ns. This is sufficient for the FPGA to compute the output signal $y(n)$, which takes approximately 15 ns.

4. RESULTS

This section presents the numerical results we obtained with our system.

4.1 Pattern Generation

The results presented in this section were obtained using the simple gradient descent algorithm (see section 2.2.2) to train the reservoir computer. This is the simplest training algorithm considered here, it provides good results, as shown below, and it has already been implemented on the FPGA board.[17,27] For these reasons, we didn't try the other two training algorithms on this task.

We started by looking for optimal values for feedback and input gain parameters α and β (see equations (3)) ensuring appropriate dynamics of the network for this task. We scanned both parameter in the most relevant intervals and recorded MSEs of the training and test sequences. Simulations were performed with 10 random patterns of length $L = 20$, the reservoir computer was trained and tested over $20k$ samples. For statistical purposes, all measures were repeated with 10 random input masks. The results are shown in figure 4(a). Each curve shows the performance as function of one parameter, while the other parameter is set to its optimal value. Blue curves correspond to the feedback gain, green curves depict the input gain. Solid and dotted lines show the error of the training and test phases, respectively. As explained in section 2.3, the training error corresponds to the MSE of the one-step ahead prediction, whereas the test error is the MSE of the reservoir computer output signal, generated autonomously, compared to the target signal. Note that feedback gain α shows a very strong influence on the reservoir computer performance and has to be set precisely at $\alpha = 1$. This provides the longest memory of the system, which is required to store the pattern.[28] On the other hand, the performance depends much less on the input gain, that is, any value in the interval $\beta \in [1.1, 1.4]$ provides sufficiently low MSEs.

We then investigated the dependence of the generator performance on the precision of the FPGA. We simulated a network with 50 neurons generating random patterns of length $L = 30$ and found that the ADC resolution has virtually no influence on the training and test MSEs. That is, setting the neuron precision to as low as 2 bits doesn't decrease performance, as long as the readout weights are computed with high precision. These results show that our ADCs and DACs are well suited for the job.

The precision of the readout weights w_i, however, plays an important role in the reservoir computer performance. We ran several simulations with 10 random patterns of length $L = 30$ and different precisions to find out the minimal resolution required for stable generation. The results are shown in figure 4(b). Solid and dotted curves show average training and test MSEs, respectively. Note that training error decreases smoothly with higher precision, while the test error curve shows more sharp behaviour. This is due to the fact that among 10 random patterns tested here, some are simple to generate even with lower precision, while others require more resolution for stable generation. That is, if the pattern is trained with enough precision, test MSE is of the same order of magnitude than the training MSE. However, if training MSE is not low enough, the error will grow up to 10^{-1} during the test phase. Following these results, we conclude that the minimal required resolution for all patterns to be generated correctly is 26 bits.

Finally, we sought for the longest pattern that could be generated with our 50-neuron reservoir computer. We discovered that patterns up to 51 elements long can be successfully generated at cost of very long training sequence of 2.5×10^6 samples. This confirms the intuitive idea that a reservoir with 50 nodes (and one bias

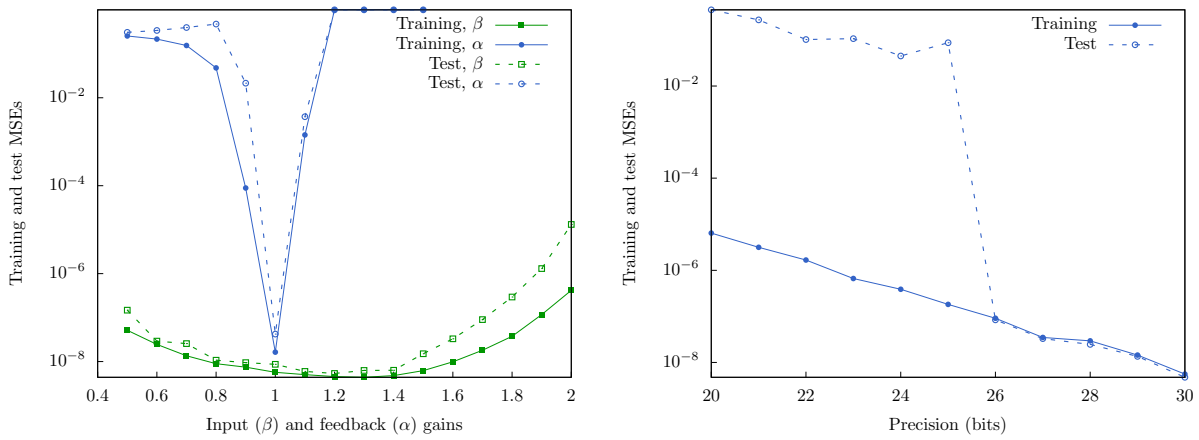


Figure 4. Results for the pattern generation task. **(a)** Optimal values of the feedback gain α (blue curves) and the input gain β (green curves) for the pattern generation task. Solid and dotted lines correspond to the mean square error of the training and test phases, respectively. **(b)** Reservoir performance for different bit-precisions of the readout weights w_i. Solid and dotted curve denote training and test errors, respectively.

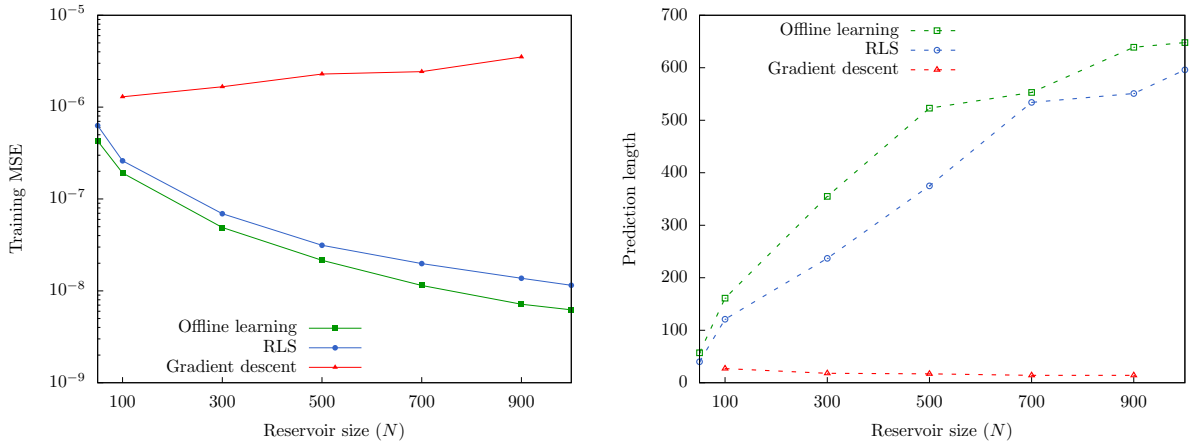
neuron, as in equation (2)) has, in theory, a linear memory of length 51, which limits the length of the pattern that can be learnt by the network. If the reservoir computer is trained to generate a pattern of length $L = 52$, even with a longer training sequence the MSE doesn't decrease below 10^{-2}.

4.2 Mackey-Glass Series Prediction

We started by investigating the choice of training algorithm and reservoir size. We measured the best performances that can be achieved with the three learning methods described in section 2.2 and different reservoir sizes, going from 50 neurons (the size of the network we were using up to now[17]) up to 1000 neurons (based on previous works on this task[1]). We compared both the MSE at the end of the training sequence (figure 5(a)) and the prediction length (figure 5(b)). Red, blue and green lines correspond to gradient descent, recursive least squares and offline learning algorithms, respectively. For statistical purposes, each measure was performed 10 times with different input masks. Training lengths were set to 5×10^5 samples for the gradient descent algorithm, 2000 for RLS and 5000 for offline learning. We didn't try to increase the training length as the goal here was not to obtain the best performance with each algorithm, but simply to find out what which methods have the potential to solve this task. The results show that both RLS and offline algorithms can be considered, while the simple gradient descent is not efficient enough. The reservoir size also plays an important role. While it is possible to predict about 50 steps ahead with a 50-neuron reservoir, much larger networks are required for longer term Mackey-Glass system emulation.

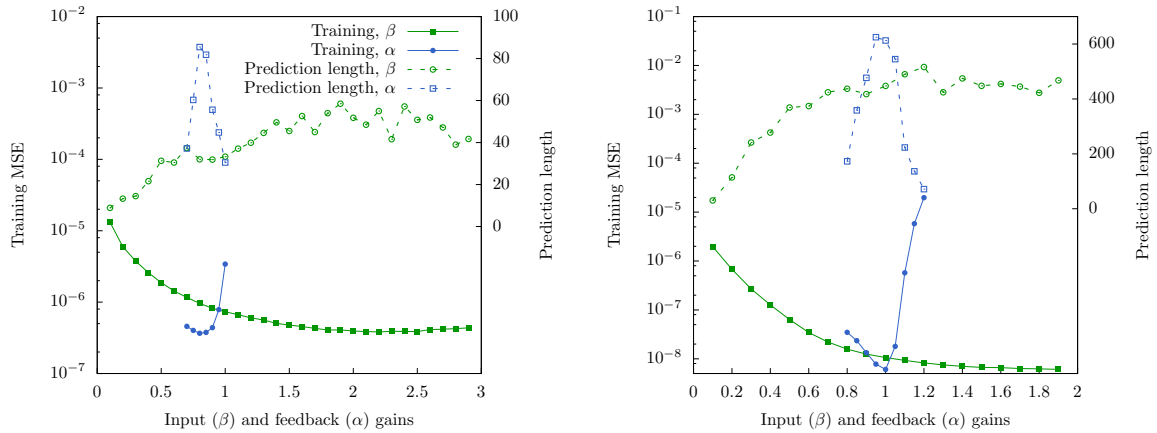
We sought for the optimal values of the input and feedback gains. We performed four simulations using the two best training algorithms, RLS and offline learning, and two reservoir sizes of 50 and 1000 neurons. During each simulation, we scanned α and β within relevant intervals. The resulting training errors (solid lines) and prediction lengths (dotted lines) are shown in figures 6(a)-6(d). Green curves correspond to the input gain β and blue ones to the feedback gain α. We note that both algorithms show similar results, which vary slightly for different reservoir sizes. The optimal input gain doesn't depend on the reservoir size and any value within $\beta \in [2, 2.5]$ interval yields low training error. The feedback parameter α needs to be tuned more precisely and depends on the reservoir size: smaller reservoirs work better with $\alpha = 0.8$, while large reservoirs require $\alpha = 1$.

Finally, we investigated the impact of computational precision on the results. We didn't consider the offline training method as it would be performed on a personal computer with sufficient floating point precision. The RLS algorithm, however, will be programmed on the FPGA chip and depends on the fixed point precision of the arithmetical operations. We ran several simulations with different precisions, different input masks (for statistical

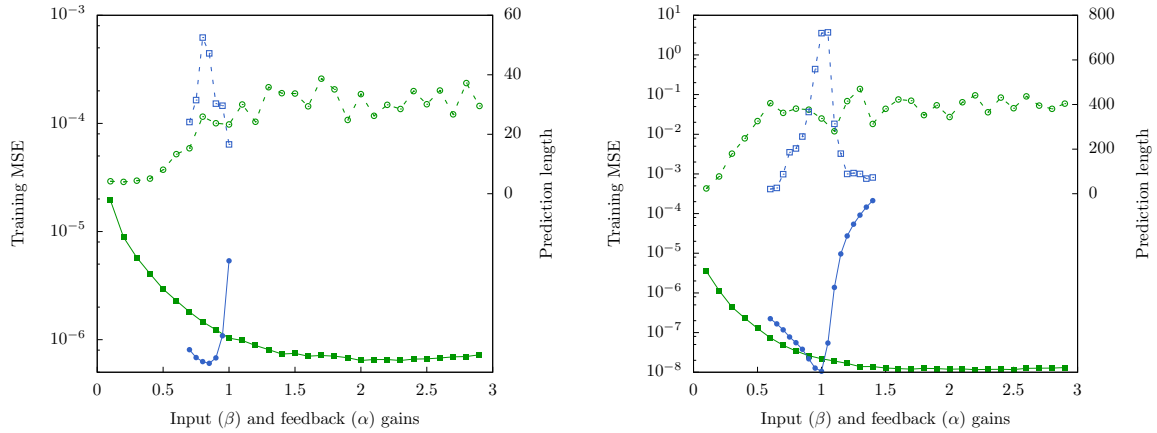


(a) (b)

Figure 5. Training errors **(a)** and prediction lengths **(b)** for different training algorithms and reservoir sizes for Mackey-Glass task.

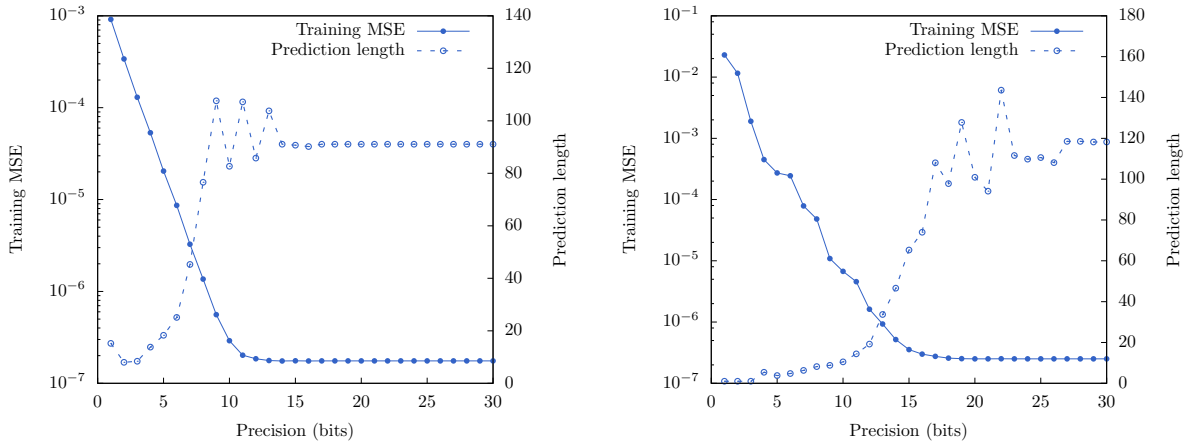


(a) Offline, $N = 50$ (b) Offline, $N = 1000$



(c) RLS, $N = 50$ (d) RLS, $N = 1000$

Figure 6. Optimal values of the feedback gain α (blue curves) and the input gain β (green curves) for the Mackey-Glass prediction task. Solid and dotted lines correspond to mean training error and prediction length, respectively.



(a) (b)

Figure 7. RLS performance for different bit-precisions of the reservoir states $x_i(n)$ **(a)** and the readout weights w_i **(b)**. Solid and dotted curve denote training errors and prediction lengths, respectively.

purposes) and $N = 100$ to find out the minimal allowed resolution. Like for pattern generation, we tested the precision of the reservoir states, sampled by the ADC (figure 7(a)), and of the readout weights, computed by the FPGA (figure 7(b)). We found that the ADC 14-bit resolution won't affect the experimental performance, as the prediction length gets stable at 14-bit precision. The computations on the FPGA board, however, need to be performed with sufficiently high precision, at least 27 bits, in order to achieve maximum prediction length.

5. CONCLUSION

Reservoir computing makes it possible to efficiently implement recurrent neural networks in hardware. Numerous optical and opto-electronic reservoir computers have been demonstrated, which is very interesting for high speed computation, data and signal processing. In the present work, we reported the first steps towards using such systems for pattern generation and chaotic series prediction. We based our work on the opto-electronic reservoir system we developed previously and showed what improvements need to be realised to this system to successfully solve these tasks.

Tackling prediction tasks in hardware has many potential applications. Precise pattern generation is required for efficient robot control. Adding the online learning makes the robot potentially capable of learning to move and adapting its motion to variable conditions. A reservoir computer can also be used as a function generator with tunable frequency,[29] or even store several different patterns and make use of the input signal to select the pattern to generate.[20,30] The Mackey-Glass task shows that a photonic device is capable of predicting chaotic series. This could be used to emulate different chaotic systems such as Lorentz or Rössler attractors. It could also be trained to make prediction from data and applied to financial forecasting, for instance. This work thus advances experimental reservoir computing towards numerous new applications.

ACKNOWLEDGMENTS

We acknowledge financial support by Interuniversity Attraction Poles program of the Belgian Science Policy Office under grant IAP P7-35 "photonics@be", by the Fonds de la Recherche Scientifique FRS-FNRS and by the Action de la Recherche Concentrée of the Académie Universitaire Wallonie-Bruxelles under grant AUWB-2012-12/17-ULB9.

REFERENCES

[1] Jaeger, H. and Haas, H., "Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication," *Science* **304**, 78–80 (2004).

[2] Maass, W., Natschläger, 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).

[3] Lukoševičius, M. and Jaeger, H., "Reservoir computing approaches to recurrent neural network training," *Comp. Sci. Rev.* **3**, 127–149 (2009).

[4] 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 (April 2009).

[5] Triefenbach, F., Jalalvand, A., Schrauwen, B., and Martens, J.-P., "Phoneme recognition with large hierarchical reservoirs," *Adv. Neural Inf. Process. Syst.* **23**, 2307–2315 (2010).

[6] "The 2006/07 forecasting competition for neural networks & computational intelligence." http://www.neural-forecasting-competition.com/NN3/ (2006). (Date of access: 21.02.2014).

[7] Appeltant, L., Soriano, M. C., Van der Sande, G., Danckaert, J., Massar, S., Dambre, J., Schrauwen, B., Mirasso, C. R., and Fischer, I., "Information processing using a single dynamical node as complex system," *Nat. Commun.* **2**, 468 (2011).

[8] Haynes, N. D., Soriano, M. C., Rosin, D. P., Fischer, I., and Gauthier, D. J., "Reservoir computing with a single time-delay autonomous boolean node," *Physical Review E* **91**(2), 020801 (2015).

[9] Paquot, Y., Duport, F., Smerieri, A., Dambre, J., Schrauwen, B., Haelterman, M., and Massar, S., "Optoelectronic reservoir computing," *Sci. Rep.* **2**, 287 (2012).

[10] Larger, L., Soriano, M., Brunner, D., Appeltant, L., Gutiérrez, J. M., Pesquera, L., Mirasso, C. R., and Fischer, I., "Photonic information processing beyond Turing: an optoelectronic implementation of reservoir computing," *Opt. Express* **20**, 3241–3249 (2012).

[11] Martinenghi, R., Rybalko, S., Jacquot, M., Chembo, Y. K., and Larger, L., "Photonic nonlinear transient computing with multiple-delay wavelength dynamics," *Phys. Rev. Let.* **108**, 244101 (2012).

[12] Duport, F., Schneider, B., Smerieri, A., Haelterman, M., and Massar, S., "All-optical reservoir computing," *Opt. Express* **20**, 22783–22795 (2012).

[13] Brunner, D., Soriano, M. C., Mirasso, C. R., and Fischer, I., "Parallel photonic information processing at gigabyte per second data rates using transient states," *Nat. Commun.* **4** (2012).

[14] Dejonckheere, A., Duport, F., Smerieri, A., Fang, L., Oudar, J.-L., Haelterman, M., and Massar, S., "All-optical reservoir computer based on saturation of absorption," *Opt. Express* **22**, 10868–10881 (2014).

[15] Vandoorne, K., Mechet, P., Van Vaerenbergh, T., Fiers, M., Morthier, G., Verstraeten, D., Schrauwen, B., Dambre, J., and Bienstman, P., "Experimental demonstration of reservoir computing on a silicon photonics chip," *Nat. Commun.* **5** (2014).

[16] Vinckier, Q., Duport, F., Smerieri, A., Vandoorne, K., Bienstman, P., Haelterman, M., and Massar, S., "High-performance photonic reservoir computer based on a coherently driven passive cavity," *Optica* **2**(5), 438–446 (2015).

[17] Antonik, P., Duport, F., Smerieri, A., Hermans, M., Haelterman, M., and Massar, S., "Online training of an opto-electronic reservoir computer," in [*APNNA's 22th International Conference on Neural Information Processing*], *LNCS* **9490**, 233–240 (2015).

[18] Ijspeert, A. J., "Central pattern generators for locomotion control in animals and robots: a review," *Neural Networks* **21**(4), 642–653 (2008).

[19] Jaeger, H., "The "echo state" approach to analysing and training recurrent neural networks - with an Erratum note," *GMD Report* **148** (2001).

[20] Sussillo, D. and Abbott, L., "Generating coherent patterns of activity from chaotic neural networks," *Neuron* **63**(4), 544 – 557 (2009).

[21] Rodan, A. and Tino, P., "Minimum complexity echo state network," *IEEE Trans. Neural Netw.* **22**, 131–144 (2011).

[22] Arfken, G. B., [*Mathematical methods for physicists*], Orlando FL: Academic Press (1985).

[23] Bishop, C. M., [*Pattern recognition and machine learning*], Springer (2006).

[24] Haykin, S., [*Adaptive filter theory*], Prentice-Hall, Upper Saddle River, New Jersey (2000).

[25] Mackey, M. C. and Glass, L., "Oscillation and chaos in physiological control systems," *Science* **197**(4300), 287–289 (1977).

[26] Seinfeld, J. H. and Lapidus, L., [*Numerical solution of ordinary differential equations*], Academic Press (1973).

[27] Antonik, P., Smerieri, A., Duport, F., Haelterman, M., and Massar, S., "FPGA implementation of reservoir computing with online learning," in [*24th Belgian-Dutch Conference on Machine Learning*], (2015). http://homepage.tudelft.nl/19j49/benelearn/papers/Paper_Antonik.pdf.

[28] White, O. L., Lee, D. D., and Sompolinsky, H., "Short-term memory in orthogonal neural networks," *Physical review letters* **92**(14), 148102 (2004).

[29] wyffels, F., Li, J., Waegeman, T., Schrauwen, B., and Jaeger, H., "Frequency modulation of large oscillatory neural networks," *Biological Cybernetics* **108**(2), 145–157 (2014).

[30] Jaeger, H., "Controlling recurrent neural networks by conceptors," *arXiv preprint arXiv:1403.3369* (2014).