The Introduction of Time-Scales in Reservoir Computing, Applied to Isolated Digits Recognition*

Benjamin Schrauwen**, Jeroen Defour, David Verstraeten, and Jan Van Campenhout

Electronics and Information Systems Department, Ghent University, Belgium
Benjamin.Schrauwen@UGent.be

Abstract. Reservoir Computing (RC) is a recent research area, in which a untrained recurrent network of nodes is used for the recognition of temporal patterns. Contrary to Recurrent Neural Networks (RNN), where the weights of the connections between the nodes are trained, only a linear output layer is trained. We will introduce three different time-scales and show that the performance and computational complexity are highly dependent on these time-scales. This is demonstrated on an isolated spoken digits task.

1 Introduction

Many real-world difficult tasks that one would like to solve using machine learning are temporal in nature. In the field of neural networks, several approaches have been proposed that introduce the notion of time into the basic concept of stateless, feed-forward networks, where the main objective is of course to give the network access to information from the past as well as the present. One well known method is to feed the signals of interest through a delay line, which is then used as input to the network (Time Delay Neural Networks). These however introduce additional parameters into the model and impose an artificial constraint on the time window. A more natural way of dealing with temporal input signals is the introduction of recurrent, delayed connections in the network, which allow the network to store information internally. These Recurrent Neural Networks (RNN) are a theoretically very powerful framework, capable of approximating arbitrary finite state automata [1] or even Turing machines [2]. Still, wide-scale deployment of these networks is hindered by the difficult and computationally costly training, caused in part by the fact that the temporal gradient information gets washed out as it is back-propagated into the past [3].

Reservoir Computing (RC) offers an intuitive methodology for using the temporal processing power of RNN without the hassle of training them. Originally introduced independently as the Liquid State Machine [4] or Echo State Networks [5], the basic concept is to randomly construct an RNN and to leave the

* This research is partially funded by FWO Flanders project G.0317.05.
** Corresponding author.

J. Marques de Sá et al. (Eds.): ICANN 2007, Part I, LNCS 4668, pp. 471–479, 2007.

weights unchanged. A separate linear regression function is trained on the response of the reservoir to the input signals using pseudo-inverse. The underlying idea is that a randomly constructed reservoir offers a complex nonlinear dynamic transformation of the input signals which allows the readout to extract the desired output using a simple linear mapping.

Evidently, the temporal nonlinear mapping done by the reservoir is of key importance for its performance. One of the appealing properties of this type of networks is the fact that they are governed by only a few global parameters. Generally, when solving a task using RC, the search for optimal reservoir dynamics is done by adjusting global scaling parameters such as the input scaling or spectral radius[1]. However, as we will show, optimizing the temporal properties of the entire system can also be very influential to the performance and the computational complexity. Since a reservoir is a dynamic system that operates at a certain time-scale, the precise adjustment of the internal temporal behavior of the reservoir to both the input signal and the desired output signal is important. In this contribution, we present an overview of the different ways in which the dynamic behavior of reservoirs has been described in literature, and we investigate the interplay between different temporal parameters for each of these models when applied to signal classification tasks.

2 Reservoir Computing and Time-Scales

Although there exist some variations on the global description of an RC system, we use this setup:

$$\begin{aligned}\mathbf{x}[t+1] &= f\left(W^{\text{res}}_{\text{res}}\mathbf{x}[t] + W^{\text{res}}_{\text{inp}}\mathbf{u}[t]\right)\\ \widehat{\mathbf{y}}[t+1] &= W^{\text{out}}_{\text{res}}\mathbf{x}[t+1] + W^{\text{out}}_{\text{inp}}\mathbf{u}[t] + W^{\text{out}}_{\text{bias}},\end{aligned}$$

with $\mathbf{u}[t]$ denoting the input, $\mathbf{x}[t+1]$ the reservoir state, $\mathbf{y}[t+1]$ the expected output, and $\widehat{\mathbf{y}}[t+1]$ the actual output[2]. All weights matrices to the reservoir ($W^{\text{res}}_{\star}$) are initialized at random, while all connections to the output ($W^{\text{out}}_{\star}$) are trained. The non-linearity f is a hyperbolic tangent.

In this system we can define three different time-scales: the time-scale of the input, the internal state, and the output. Traditionally these are all the same, but in this paper we will show that performance can be improved and that computational demands can be decreased by setting these time-scales correctly.

2.1 Input Time-Scale and Integrator Nodes

In [6], the notion of input time-scales and the link to node integration was introduced. In this contribution we will look at three ways to add an integrator to a node: after the non-linearity (as used in [6]), before the non-linearity (as

[1] The largest absolute eigenvalue of the connection matrix.

[2] We denote discrete time with $[t]$ and continuous time with (t).

used in continuous time RNN), and over the non-linearity (which we introduce). We will first discuss the integrator after the non-linearity.

The input time-scale and integrator can be introduced by starting from a continuous time differential equation:

$$\dot{\mathbf{x}} = \frac{1}{c}\left(-a\mathbf{x} + f\left(W_{\text{inp}}^{\text{res}}\mathbf{u} + W_{\text{res}}^{\text{res}}\mathbf{x}\right)\right)$$

where a denotes the retainment rate (which in this work we set to 1, meaning that no information is leaked), and c is a scaling factor for the temporal dynamics. If this is discretely approximated by using Euler discretization we get:

$$\mathbf{x}((t+1)\delta_2) = (1-\lambda)\mathbf{x}(t\delta_2) + \lambda f\left(W_{\text{inp}}^{\text{res}}\mathbf{u}(t\delta_2) + W_{\text{res}}^{\text{res}}\mathbf{x}(t\delta_2)\right)$$

where δ_2 is the Euler step and $\lambda = \delta_2/c$. Note that the Euler time step δ_2 determines the sample rate of the input, while c can be used to scale the speed of the dynamics.

Although the retainment rate was the major research topic of [6] and λ was ignored, this work focuses on λ and sets the retainment rate to 1. This is because that parameter was previously not thoroughly investigated, and when changing λ the effective spectral radius of the reservoir does not change, while, when changing a, it does. When changing the spectral radius, the dynamic regime of the reservoir, which is very important for the performance of the reservoir, changes. This coupling between time-scale settings and dynamic regime settings is not desired.

When the integrator is placed before the non-linearity, which is common practice in continuous time RNN, we end up with these equations:

$$\begin{aligned}\mathbf{z}((t+1)\delta_2) &= (1-\lambda)\mathbf{z}(t\delta_2) + \lambda\left(W_{\text{inp}}^{\text{res}}\mathbf{u}(t\delta_2) + W_{\text{res}}^{\text{res}}\mathbf{x}(t\delta_2)\right)\\ \mathbf{x}((t+1)\delta_2) &= f(\mathbf{z}((t+1)\delta_2)).\end{aligned}$$

This has the advantage of being stable: if an integrator starts to blow up, it is constrained by the non-linearity, which is not the case in the previous model.

Yet another, empirical model, is possible by placing the integrator *over* the non-linearity:

$$\mathbf{x}((t+1)\delta_2) = f\left((1-\lambda)\mathbf{x}(t\delta_2) + \lambda\left(W_{\text{inp}}^{\text{res}}\mathbf{u}(t\delta_2) + W_{\text{res}}^{\text{res}}\mathbf{x}(t\delta_2)\right)\right).$$

The advantage of this is that an integrator can never blow up and that it has no computational overhead since the integrators can be incorporated in the W matrix by increasing the diagonal elements. But a drawback is that the integrator does leak away even with $a = 1$. The leak is due to the contracting property of the non-linear mapping of the hyperbolic tangent upon itself. This has as a consequence that the overall amplitude of the reservoir dynamics scales down when λ goes to 0.

Theoretically there are some strong and weak points concerning these three possible models, but we will have to experimentally investigate which of these is somehow 'optimal'.

The input of the reservoir is in practice always a sampled signal. We denote the sampling time-scale of the input δ_1. The resampling factor from input to internal time-scale is thus equal to δ_2/δ_1. This resampling should approximate the ideal Nyquist, non-aliasing resampling operation. We use the Matlab `resample` operation for this.

2.2 Output Time-Scale

In this work we also introduce an output time-scale

$$\widehat{\mathbf{y}}((t+1)\delta_3) = W_{\text{res}}^{\text{out}}\mathbf{x}((t+1)\delta_3) + W_{\text{inp}}^{\text{out}}\mathbf{u}(t\delta_3) + W_{\text{bias}}^{\text{out}}.$$

The reservoir states and inputs are thus resampled to this new time-scale before training and applying the linear readout. The resampling factor from internal to output is δ_3/δ_2. We will call this reservoir resampling.

At a first glance this time-scale does not seem to be very important, but as we will show in the experiments, changing this time-scale can have a drastic effect on performance.

3 Experimental Setup and Results

For all of the following experiments, we used the Reservoir Computing Toolbox[3] [7], which allows us to do all the simulations in the Matlab environment. With this toolbox, we will study the task of speech recognition of isolated digits. The Lyon passive ear model [8] is used to frequency-convert the spoken digits into 77 frequency channels. The task of recognizing isolated spoken digits by RC has already been studied [7], and the results have been very positive. The dataset we will use (a subset of the TI48 dataset) contains 500 spoken isolated digits, the digits 0 to 9, where every digit is spoken 10 times by 5 female speakers. We can evaluate the performance of the reservoir by calculating the Word Error Rate (WER), which is the fraction of incorrectly classified words as a percentage of the total number of presented words. Because of the randomness of the reservoirs, we will do every simulation 20 times with different stochastically generated reservoirs, and use 10-fold cross validation to obtain a decent statistical result.

Ridge regression (least squares optimization where the norm of the weights is added as a penalty) was used to train the readout, where the regularization parameter was set by doing a grid search. To classify the ten different digits, ten outputs are trained which should be 1 when the digit is uttered, -1 if not. The temporal mean of the ten classifiers is taken over the complete sample, and a Winner-Take-All is applied to this. The winner output represents the classified digit.

The settings of the reservoir, like reservoir size which is 200 and spectral radius which is 0.7, are in this paper intentionally chosen to be non-optimal so

[3] Which is an open-source Matlab toolbox for Reservoir Computing, freely available at http://www.elis.ugent.be/rct .

Table 1. Minimal classification errors for all cases

	input resampling	reservoir resampling
integrator after non-linearity	2.15%	0.48%
integrator before non-linearity	2.32%	0.48%
integrator over non-linearity	**1.36%**	**0.40%**

we still have a margin of error left to evaluate the resampling techniques on. If everything is set optimally we can very easily get zero training error!

In [6] the intricate interplay between optimal values of a and the spectral radius is investigated. When changing λ, we notices that there is no such intricate dependency to the spectral radius. The optimal spectral radius is not very dependent on the value of λ.

3.1 Input Resampling vs. Integration

First we will study the influence of input resampling versus integration for the different integrator options on the performance of the reservoir. To be able to handle a broad range of parameters, we vary both these parameters in a logarithmic way, with base 10. Figure 1 shows the results of this experiment.

For these experiments, the internal time-scale δ_2 is set to 1 and the input time-scale δ_1 is changed. This is similar to keeping δ_1 constant and changing δ_2. Note that when keeping λ constant and changing δ_2, c changes. We can observe a diagonal area on Figure 1 which corresponds to an optimal performance. This optimal performance is actually achieved with a fixed value of c, but we perceive it as a λ value which changes linearly with δ_1, but this is due to the fact that $\log_{10}(\lambda) = \log_{10}(\delta_2)/\log_{10}(c)$.

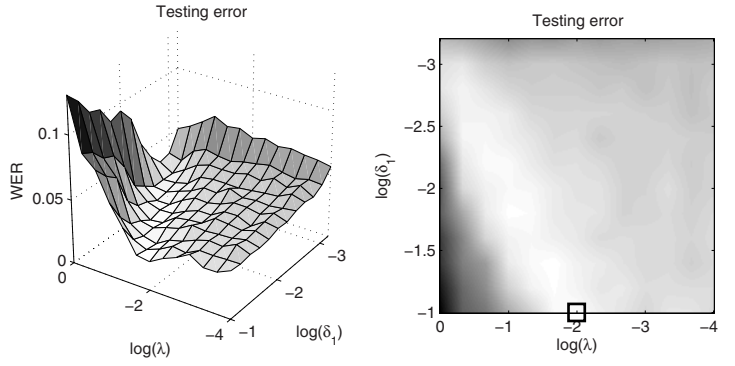
For all three integrator cases we see that the optimal performance is attained with the least resampling (bottom of the plots). However, if we resample more, the error only slightly increases. This creates a trade-off between computational complexity and performance.

The optimal performance for the three different integrator settings are given in Table 1. We see that the integrator over the non-linearity performs optimally, which is nice, because this introduces no extra computational requirements.

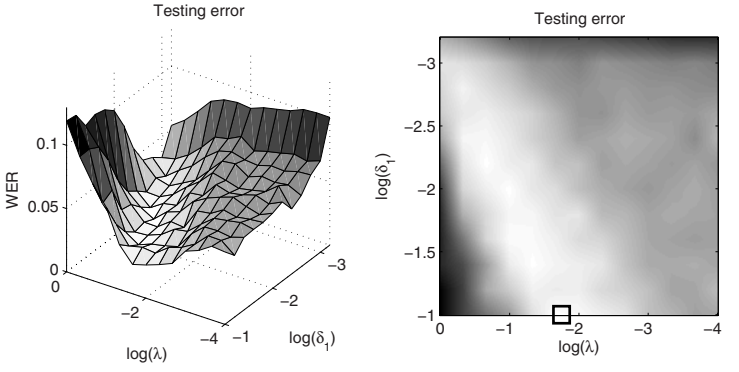
3.2 Reservoir Resampling vs. Integration

The second experiment studies the output time-scale compared to the internal integrator. The results of this experiment are shown in Figure 2. For these experiments, the input time-scale is set to $\log_{10}(\delta_1) = -2$ and the internal time-scale $\delta_1 = 1$. The setting of the input time-scale is not critical since the conclusions of the results also apply to other input time-scale settings.

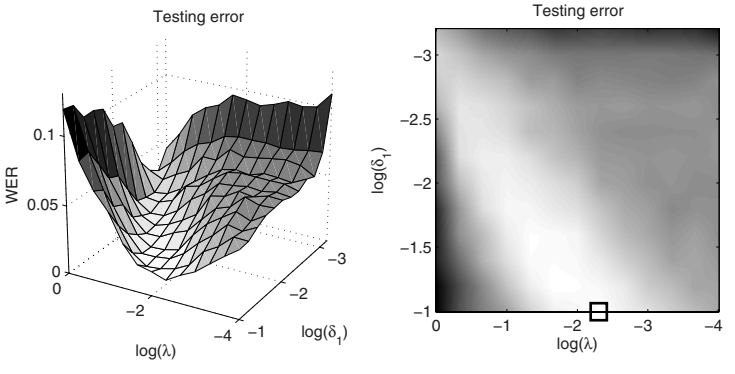
These figures are a bit more complex to interpret. The upper part, with $\log_{10}(\delta_3) = 0$, has no reservoir resampling and is thus equal to a slice of Figure 1 where $\log_{10}(\delta_1) = -2$. When increasing the resampling of the reservoir states, we see that for the region of low integration (close to 0) there is a significant



(a) integrator after the non-linearity

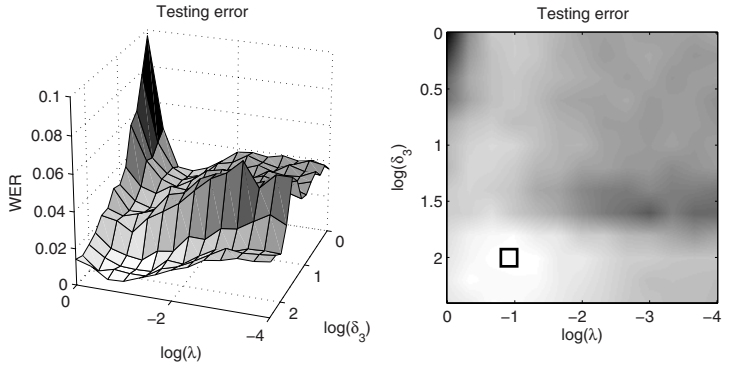


(b) integrator before the non-linearity

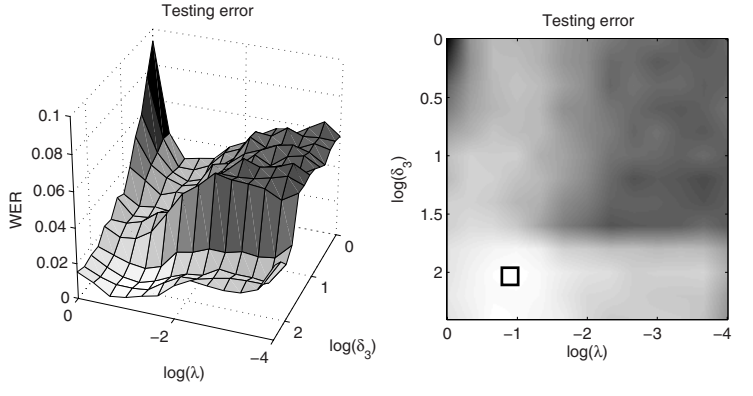


(c) integrator over the non-linearity

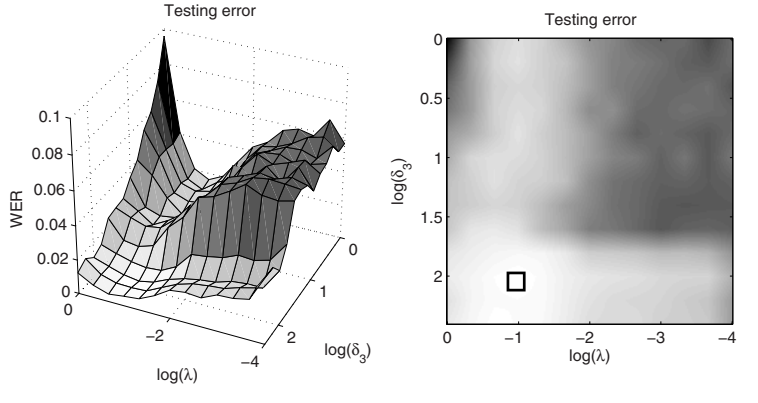
Fig. 1. WER results for input resampling versus integration for the three integrator options. The small squares denote minimal error.



(a) integrator after the non-linearity



(b) integrator before the non-linearity



(c) integrator over the non-linearity

Fig. 2. WER results for reservoir resampling versus integration for the three integrator options. The small squares denote minimal error.

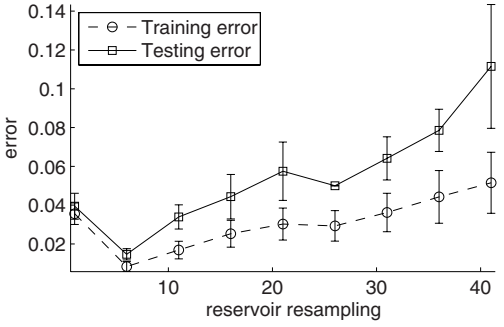


Fig. 3. Classification error for a signal classification task with respect to the output time-scale. Note that the time-scale is not logarithmic in this figure.

decrease of the error. But in the region where the integration is optimal there is initially no improvement.

The bottom part of the figures show a drastic drop in performance when $\log_{10}(\delta_3)$ is larger than 1.5. With such a high input and reservoir resampling, there is actually just one time step left! For this task we thus have optimal performance when reducing all the reservoir's dynamics to a single point in state space: the centroid of the dynamics in state space. This is not completely awkward since the post-processing of the linear classifier's output is anyway by taking its temporal mean before applying Winner-Take-All. The major drawback of this drastic reservoir resampling is that all temporal information is lost. With less reservoir resampling, the reservoir is able to already give a prediction of the uttered word even if it is for example only partially uttered. We thus trade-off performance to the ability of on-line computation.

One might think that when averaging out all the reservoir's dynamics, it has no real purpose. But when training a linear classifier to operate on the temporal average of the frequency-transformed input, so without using a reservoir, we end up with an error of 3%. This is quite good, but still an order of a magnitude worse than when using a reservoir of only 200 neurons.

These conclusions are, however, partly due to the fact that here the desired class output remains constant during the whole input signal. Figure 3 shows that this is not generally the case. Here, the task is to do signal classification, whereby the input signal is constructed by concatenating short signal pieces of 50 time steps, whereby every piece is either a noisy sawtooth or a noisy square wave with the same period. The task is then to classify the signal type at every time step. As the results show, in this case there is a trade-off between the amount of resampling of the reservoir responses and the amount of information available to the linear readout to do the classification. In this case only a small amount of reservoir resampling is needed to attain optimal performance. In state space this can be seen as taking the centroid of a small temporal region of the trajectory. This temporal averaging out of the dynamics seems to significantly increase the classification performance of the RC system.

4 Conclusion and Future Work

It was previously already mentioned that setting input time-scales is possible [6], however, in this work we show that setting this time-scale is critical for getting optimal performance. We showed that there is a clear link between node integration and resampling. When using input resampling the computational complexity decreases linearly with resampling, while only slightly influencing the performance. We introduced three types of node integration and showed how they perform on an isolated spoken digits classification task. The integrator scheme introduced in this work performs optimally and can be implemented without overhead.

Next we showed that an output time-scale can also be introduced. Although that initially this might not seem to be influential on performance, we experimentally show that a large performance gain can be achieved by optimally setting this resampling. Here we get a speed-up and an improvement in performance. This result suggests that for classification tasks it is not the actual precise dynamics that are important, but more the temporally filtered dynamics which is the local centroid of the dynamics in state space.

There are many other options for future research based on this work. We will investigate how this resampling scheme is applicable to more complex tasks like continuous speech, where classification needs to be performed on-line, and where there are multiple time-scales presents. To solve this we might need to introduce hierarchical or heterogeneous reservoirs with different parts working at different time-scales. Ultimately these temporal modules should be created autonomously.

References

1. Omlin, C.W., Giles, C.L.: Constructing deterministic finite-state automata in sparse recurrent neural networks. In: IEEE International Conference on Neural Networks (ICNN'94), Piscataway, NJ, pp. 1732–1737. IEEE Computer Society Press, Los Alamitos (1994)
2. Kilian, J., Siegelmann, H.T.: The dynamic universality of sigmoidal neural networks. Information and Computation 128, 48–56 (1996)
3. Hammer, B., Steil, J.J.: Perspectives on learning with recurrent neural networks. In: Proceedings of ESANN (2002)
4. Maass, W., Natschläger, T., Markram, H.: Real-time computing without stable states: A new framework for neural computation based on perturbations. Neural computation 14(11), 2531–2560 (2002)
5. Jaeger, H.: Short term memory in echo state networks. Technical Report GMD Report 152, German National Research Center for Information Technology (2001)
6. Jaeger, H., Lukosevicius, M., Popovici, D.: Optimization and applications of echo state networks with leaky integrator neurons. Neural Networks (to appear, 2007)
7. Verstraeten, D., Schrauwen, B., D'Haene, M., Stroobandt, D.: A unifying comparison of reservoir computing methods. Neural Networks (to appear, 2007)
8. Lyon, R.: A computational model of filtering, detection and compression in the cochlea. In: Proceedings of the IEEE ICASSP, pp. 1282–1285. IEEE Computer Society Press, Los Alamitos (1982)