Universitat
de les Illes Balears

Multilayer reservoir computing to overcome the memory-nonlinearity trade-off

Samuel Jaume Suárez

Master's Thesis

Master's degree in Physics of Complex Systems
at the
UNIVERSITAT DE LES ILLES BALEARS

Academic year 2018/2019

16 September, 2019

UIB Master's Thesis Supervisor: Miguel Cornelles Soriano

Acknowledgements

First of all, I would like to express my gratitude to my supervisor Miguel for his constant availability and patience to guide me in this field that was completely unknown for me a few months ago.

And last but not least, I would like to thank my family all the unconditional support that I have received, not only along this thesis, but also along all my academic path. Without them, simply I would not be at this point in this moment.

Abstract

Machine learning, and more precisely reservoir computing, is a state-of-the-art research field because of its successful application to time-dependent computationally-hard tasks.
Along this MSc Thesis, we will extend the basic concepts and features of Echo State Networks, which are a specific type of reservoir computers.
The main goal of this work will be finding out a reservoir configuration on which the so-called memory-nonlinearity trade-off is overcome.
In order to achieve such an enhancement, we will explore changes in the main properties of the reservoir, including its parameters, its degree of linearity and its topology.

Contents

Chapter 1

Introduction

Along this chapter, we will introduce the aim of this work and, mainly, what exactly is reservoir computing (hereafter abbreviated as RC). We will illustrate how RC works, its typical applications and in which tasks we will implement it.
The leitmotif of this master thesis will be trying to achieve a regime on which the system is able to perform simultaneously and successfully all the considered tasks. This could seem easy at first sight, but the the specific tasks in question are potentially incompatible between them. Specifically, one of the tasks practically only requires from the linear memory of the system, while other tasks require the most from the nonlinear information processing capacity of the system. However, when linear memory is improved, nonlinear information processing capacity typically decreases.
Thus, our purpose will be modifying the features of the reservoir in order to find out the coexistence of optimum linear memory and nonlinearity.

1.1 Introduction to reservoir computing

Since the beginning of the previous decade, reservoir computing has been a widely used technique for tasks that require recurrent neural networks (i.e. neural networks on which the propagating signal passes more than once in the same neuron [1],[2],[3]). But before starting introducing the many applications that RC has, it is convenient to explain how RC works and its main components.

In a general way, we call reservoir computers to the dynamical systems that process an input, in most of the cases with nonlinear nodes, and are trained for a certain task with a readout function [4],[5],[6]. There are many types of RC implementations, but the main ones are Liquid State Machines (LSM) and Echo State Networks (ESN).
On the one hand, LSM are typically used for spiking neural networks implementations (which generally try to reproduce a natural neural network for speech recognition and computer vision tasks); their main characteristics are that each node of the liquid or reservoir receives a continuous input, the connection between nodes is random and an internal state is generated, which includes the current response in the context to preceding perturbations [3].

On the other hand, ESN are generally used for time-discrete neural networks and their main characteristic are the echo states, which means that their current reservoir state depends on the previous reservoir states.
These RC implementations have the so-called "fading memory", which will be the responsible of having information of the past input points at the present time.
Along this project, we will use a subtype of ESN which is called delay-based reservoir computer. In our case, instead of having random connections between real nodes, we will have a ring of virtual nodes temporally ordered. It has to be clarified that, in our simulations, there will not be any difference between virtual and real nodes. Delay-based RC was introduced to facilitate experimental implementations of this computing paradigm.
In the following figure, a general ESN and a delay-based RC are represented:

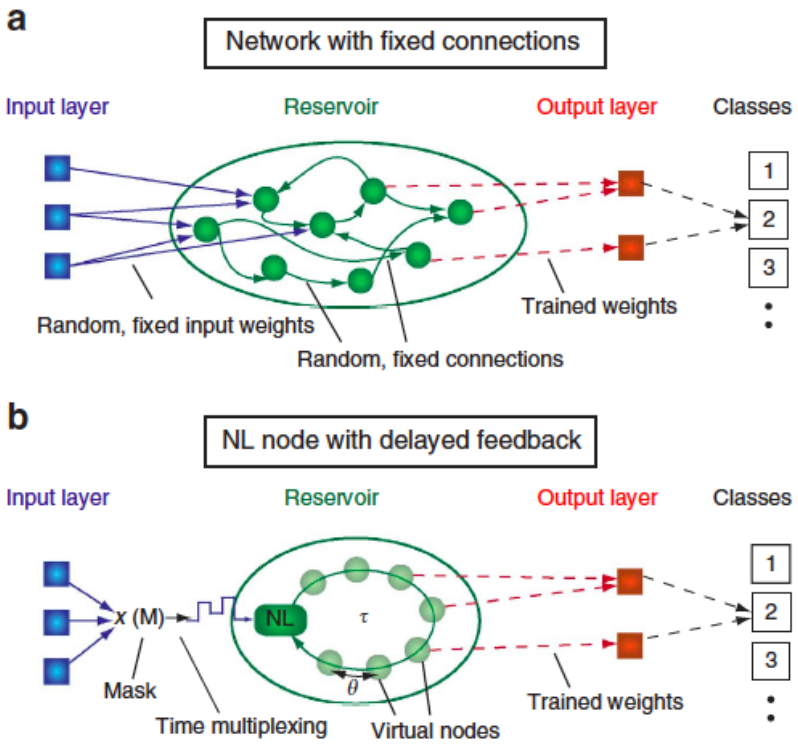

Figure 1.1: (a) Scheme of a traditional ESN. (b) ESN delay-based scheme. Figure taken from [7].

From Figure 1.1, we can appreciate the general features of RC: an input layer that is connected to the nonlinear nodes with fixed and random input weights, a reservoir that is composed by a set of real nodes and an output layer on which it will be applied the readout function.
If we look now into the details of sketch (b) of Figure 1.1 (which will be our main configuration from now on), we can see how the random fixed input weights are replaced by a temporal mask. This mask will be, as a matter of fact, a set of random fixed input weights too, but it arises from the need to associate a temporal value to each virtual node in the reservoir.
Each value of the input signal will be introduced to the reservoir every τ interval of time. Thus, we will use the mask to split the input into a set of numbers (temporally

ordered and separated by a time subinterval θ) with the same dimension than the reservoir (i.e. the mask will have the same number of weights than the number of virtual nodes in the reservoir).
In order to clarify that virtual nodes and the mask comes from the need of having a temporal processing of the input, we show the following scheme that illustrates how the input is pre-processed before arriving to the nonlinear node:

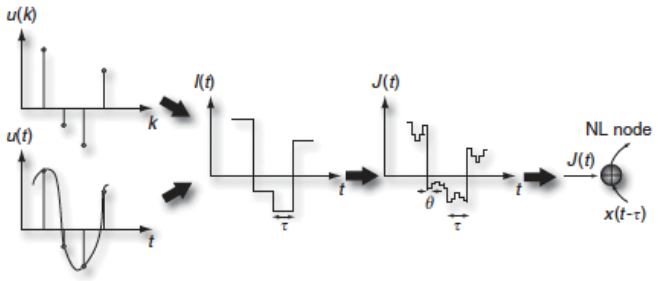

Figure 1.2: It can be seen how a continuous signal $u(t)$ is discretized as $u(k)$ and transformed as a continuous staggered signal $I(t)$. Then, when we multiply $I(t)$ by the mask M, we get the temporal input sequence $J(t) = M \cdot I(t)$. Figure taken from [7].

Now that we have explained the input pre-processing of the delay-based reservoir computer, we are going into the details of the performance of the reservoir.
In order to achieve the desired behavior of the reservoir, there are some features that must be accomplished.
The first one was already mentioned earlier; it needs to have fading memory (i.e. the reservoir current state must depend on the immediate past states but it must not depend on the far past states).
Secondly, the reservoir state has to respond differently between two different inputs but it must not be too sensitive from small variations in the input, it has to provide the same result in this case [7]. This property is related to the consistency properties of the reservoir [8].
And finally, the reservoir is designed to do demanding tasks, thus we will need a nonlinear mapping that is explained below.

The nonlinear node in delay-based RC, is an artificial neuron that acts as a nonlinear map of the input stream. Most precisely, the reservoir maps the input into a higher dimensional space corresponding to the transient response of the reservoir, where the signal is represented [7]. This idea can be seen more clearly in Fig. 1.3, where we are only dealing with a few dimensions.

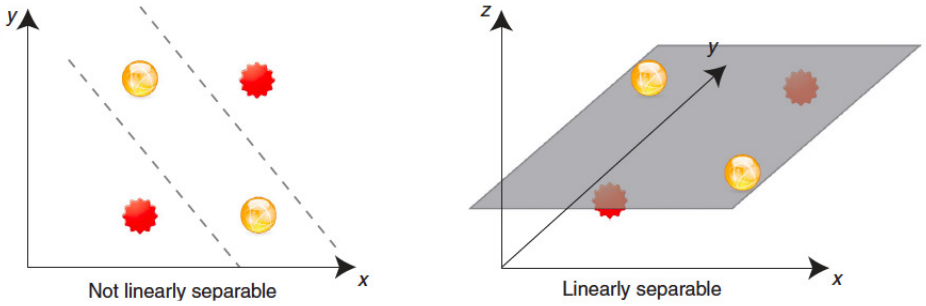

Figure 1.3: In the figure from the left we can appreciate that it does not exist any linear map or unique linear transformation that separates yellow balls from red stars (we need at least two lines); whereas in the figure from the right, introducing an extra dimension with the nonlinear map, we achieve the desired linear separability. Figure taken from [7].

Typically, the larger the number of virtual nodes is (or the larger dimension of the space of the reservoir is), the more likely is to find a linear separation [10]. For this reason, we will work with reservoirs with hundreds of virtual nodes.
In the next section, we will define the precise nonlinear transformation in order to allow the reservoir to perform computationally-hard tasks.

The output layer, will be explained in detail in section 1.3. Below, we are going to introduce the main applications that ESN have.

An important feature of RC is its classification capacity. For this reason, ESN is a state-of-the-art technique in all kind of tasks that require pattern recognition, as for example: speech recognition, optical character recognition, robotics (on which after a sensory input it is expected a movement)... [1].
Despite the amount of different applications that this RC technique has, for us, one of the most important applications will be the chaotic time series prediction, as we will explain in section 1.4.

RC is an appealing technique for all the applications mentioned above also because of its optimal hardware implementation. These implementations present high processing speed, parallelism and low power consumption [4].
However, as it could be expected, hardware-based RC has some limitations. The main ones are the noise sensitivity of the physical analog systems and the non-instantaneous response of the system to an input signal. This last inconvenient appears when θ (time spacing between virtual nodes) is smaller than the response time of the system. In this case, computation capacity decreases because the system dynamics couples consecutive virtual nodes [4].
But, all in all, it is proven that these limitations can be, to some extent, mitigated and are justified by the wide benefits that these different RC techniques provide.

1.2 Objective & nonlinear map

As it has been mentioned at the beginning of the chapter, our main objective along this work will be finding out a system configuration on which it is maximized the computation and memory capacity of the reservoir. For this reason, we will consider tasks as short-term memory STM (the one that requires from memory capacity) and parity check PC (the one that requires nonlinear computation capacity), but this will be detailed in section 1.4.

In order to achieve our objective, we will have to modify the reservoir in its topology and composition in such a way that it will be explained in further sections. But, furthermore, we will have to modify the parameters that determine the reservoir performance and its properties; those related with the nonlinear map chosen that will be introduced below.

In a general way, an ESN is determined by the following equations [11]:

$$\mathbf{r}(n) = F\left(\gamma \mathbf{W}^{\text{in}}\mathbf{x}(n) + \beta \mathbf{W}\mathbf{r}(n-1)\right) \tag{1.1}$$

$$\mathbf{o}(n) = \mathbf{W}^{\text{out}}\mathbf{r}(n) \tag{1.2}$$

In Eq. (1.1) we have the basic ingredients to build an ESN. First of all, $\mathbf{r}(n)$ is the reservoir vector or state corresponding to the n-th point of our time series (also named the n-th example), hence $\mathbf{r}(n-1)$ is the reservoir state of the previous point. We express in a general way the input $\mathbf{x}(n)$ as a vector, but in our case it will be always a scalar.
In turn, $\mathbf{W}^{\text{in}}$ corresponds to the previously mentioned mask (i.e. the input matrix, which it will be composed by random gaussian numbers between $[-1, 1]$), whereas $\mathbf{W}$ is the so-called reservoir network connectivity matrix. Moreover, γ and β are the input and connectivity scaling factors, respectively, and the function F is the nonlinear map or nonlinear activation function.

Now, if we look to Eq. (1.2) we can observe the general output vector $\mathbf{o}(n)$, which for us it will be a scalar, and the output weights matrix $\mathbf{W}^{\text{out}}$, whose training will be the key point for the reservoir performance.
It is important to notice that a few years ago, it has been proven that a reservoir with a simple topology (as it could be a ring, for example) indeed performs practically equal than a randomly connected reservoir [12]. Taking this into account, we can rewrite Eq. (1.1) in the following way disregarding the connectivity matrix $\mathbf{W}$ and building a ring topology:

$$r_i(n) = F\left(\gamma \mathbf{W}_i^{\text{in}}\mathbf{x}(n) + \beta r_{i-1}(n-1)\right), \quad i = 2...D \tag{1.3}$$

$$r_1(n) = F\left(\gamma \mathbf{W}_1^{\text{in}}\mathbf{x}(n) + \beta r_D(n-1)\right) \tag{1.4}$$

Where D is the total number of virtual nodes and the subscript i refers to the i-th node. In the end, we will have a collection of reservoir states $\mathbf{r}(n)$ in form of a big matrix that will have a dimension $n \times D$ (remembering that n is the number of examples of the time series). This will be important for the training.

Taking all these general expressions into account, now we are going to finally specify the nonlinear map F. In order to maintain a connection to potential optoelectronic implementations, we will choose the following $\sin^2$ expression which reproduces a Mach-Zender modulator (MZM) [11]:

$$r_i(n) = \sin^2\left(\phi + \gamma W_i^{\text{in}} x(n) + \beta r_{i-1}(n-1)\right) \tag{1.5}$$

Here, ϕ is the MZM offset phase. Thus, the parameters to modify in order to explore the performance of the reservoir will be ϕ, γ and β. We note that the ring connectivity in Eq. (1.2) is typical of delay-based RC.

1.3 Training & overfitting

Now, we will detail what happens in the output layer of the reservoir computer. We will explain how the output weights are determined in the so-called training process.

If we want to train our reservoir for a certain task, it is a must having a considerable sample of points (or examples) in our time series. In all our tasks we will work with time series with 4000 points, in which 3000 of them will be used to determine the output weights. Although there is no fixed rule that determines how many points we need to perform the training, one always needs a number of examples sufficiently bigger than the number of nodes D.

Hence, after processing with the reservoir the 3000 first points of our time series (hereafter called as training points), we will build a results matrix ($\mathbf{R}$) of dimension $3000 \times D$ (where D is the number of virtual nodes of the reservoir). If we want to avoid the influence of transients, we will have to disregard some of the first rows of the results matrix corresponding to the first examples (typically, we will disregard less than 100 rows).

Taking this into account, if we had the weights vector, the output results of the training points will be computed as follows (being $\mathbf{R}$ the results matrix of the reservoir):

$$\mathbf{o} = \mathbf{R}\mathbf{W}^{\text{out}} \tag{1.6}$$

The aim of the training is to determine the weights vector $\mathbf{W}^{\text{out}}$, from the examples. For this reason, we will substitute in the previous equation the vector $\mathbf{o}$ for the known examples vector $\mathbf{y}$ that corresponds to the results that we want the reservoir to provide for each different input. Thus, what we will have to do is to minimize the mean square error $\left|\left|\mathbf{R}\mathbf{W}^{\text{out}} - \mathbf{y}\right|\right|^2$ in order to obtain a general vector of fitted weights that performs the desired task for every possible input.

There is a final technical problem that we have to take into account and it is called overfitting. If we do not impose some limitations on the fitting of the output weights, these ones will become too dependent on the training data and the reservoir will not perform suitably when the remaining values of the time series will be introduced out of the training stage. For this reason, we will introduce a penalization or regularization parameter λ (which will be $\lambda = 10^{-10}$ in almost all the cases) and, instead of performing a typical minimization of the mean square error, we will actually proceed with a ridge regression, which consists in minimizing $\left|\left|\mathbf{R}\mathbf{W}^{\text{out}} - \mathbf{y}\right|\right|^2 + \left|\left|\lambda\mathbf{W}^{\text{out}}\right|\right|^2$ [1].

At this point, we have covered and explained all the reservoir stages. Thus, we will only have to select a task, determine its correspondent output weights and finally use the reservoir to perform the desired task with the previously trained weights.

The complete operation of the RC is sketched in the following figure:

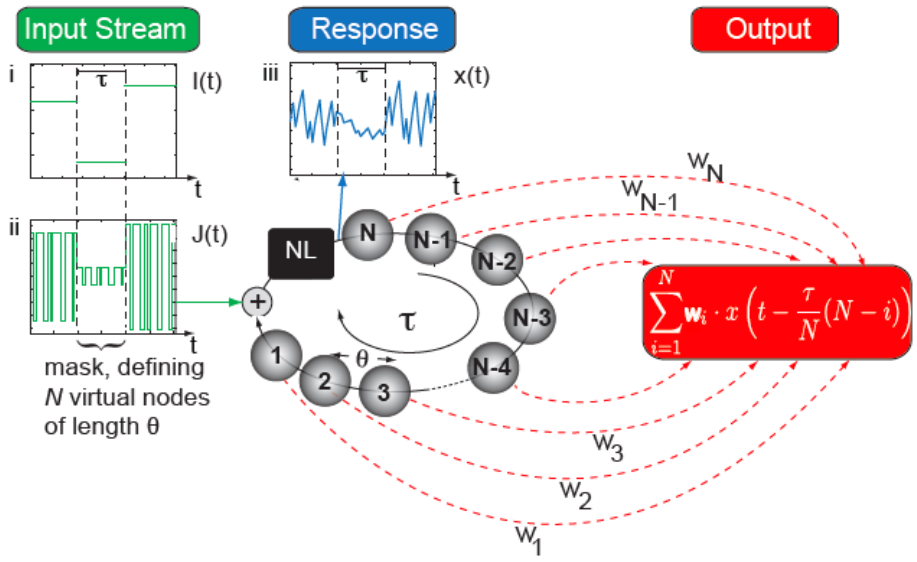

Figure 1.4: Sketch of the general procedure of the reservoir. In the first stage we have the input stream and the masking process (being a binary mask for illustration puroposes), then at the second stage we have the dynamical evolution of the reservoir and finally in the last stage we have the weighted response that will correspond to the desired output of the selected task. Figure taken from [1].

1.4 Tasks to perform

The first task to be considered will be the Short Term Memory verification (hereafter abbreviated as STM). In this task, we will generate a set of 4000 gaussian random numbers between $[-1, 1]$. Then, we will train the reservoir to determine which is the p-th number that precedes the given input. Expressed in a mathematical way (being x the input and y the target output):

$$y_i = x_{i-p} \tag{1.7}$$

As it could be figured out, the bigger p is (i.e. the further away in the past we go) the worse performance will the reservoir exhibit. For this reason, we will explore the reservoir parameters and its composition to increase as much as possible the fading memory of the system.

The second task will consist in a Parity Check (hereafter abbreviated as PC) of the sum of the p previous inputs.
Focusing on the details, we will have a 4000 point random series, whose values can be either 0 or 1. Then, we will choose how many past steps p do we want to consider, we will sum the numbers from the past p to the present and evaluate its parity; all these concepts are expressed with the following equations [14]:

$$y_i = Q\left(\sum_{m=1}^{p} x_{i-m}\right) \tag{1.8}$$

$$Q(x) = \begin{cases} 0 & [x \equiv 0(\text{mod}2)] \\ 1 & (\text{otherwise}) \end{cases} \tag{1.9}$$

As we can see, STM and PC require memory to be executed, but PC requires in addition nonlinearity because the task itself is a nonlinear map. For this reason, they will be complementary tasks; while STM will mainly evaluate the linear memory capacity of the system, PC will be useful to evaluate the capacity of the system to perform nonlinear tasks [14]. In general, it will be hard to achieve a regime on which both of these features will be enhanced simultaneously. When the system has a considerable nonlinear capacity, it typically has no linear memory at all and vice versa (this is known as the memory-nonlinearity trade-off [15] and it will be explained in more depth in section 1.6).

Finally, the last task that we will deal with is the prediction of a chaotic time series, most precisely, we will try to predict a Mackey-Glass series of 4000 points [13]. Going into details, the prediction task will consist in, given an input, provide the following value of the temporal series. Thus, the objective or target vector $\mathbf{y}$ of the training process will be exactly the input vector, but displaced one step to the future in the time series.

Moreover, in this task, we will consider the RMSE in order to quantify the error between the real series and our obtained results. It is defined as follows (being **o** the obtained result, **y** the target value and $\langle\ \rangle_n$ the mean over all the considered points that are not used in the weights determination stage):

$$RMSE = \sqrt{\langle (o-y)^2 \rangle_n} \tag{1.10}$$

Although STM and PC will be our main two tasks, we will look for a simultaneously good performance of the reservoir for the prediction task too. As we will appreciate at the second chapter, when STM is enhanced, usually the prediction task is also performing well.

1.5 Memory Capacity

Along the project, the main property that will be assessed to analyze the fading memory of the system will be the so-called memory capacity.
To express mathematically the memory capacity, we have to previously define the memory function $m(t)$, which, in fact, is the normalized linear correlation between the desired outputs $y(n)$ and their associated predicted outputs considering the past time "t" $o_t(n)$ [11]:

$$m(t) = \mathrm{corr}[o(n), y(n)] = \frac{\langle y(n) o_t(n) \rangle_n^2}{\sigma^2(y(n))\sigma^2(o_t(n))} \tag{1.11}$$

Where $\langle\ \rangle_n$ is the mean over all the test points of our temporal series (i.e. those ones that are not used for the training stage) and σ^2 is the variance of the target output $y(n)$ and the obtained output $o_t(n)$.

Then, considering the definition of $m(t)$, the memory capacity (MC) is computed as the sum of the memory function when t goes to infinity (i.e. when we consider infinite past times) [11]:

$$MC = \sum_{t=1}^{\infty} m(t) \tag{1.12}$$

Although at first sight it could seem that MC is only useful to study the linear memory of the system, in fact, it will be essential to evaluate the nonlinear memory of the system. When we apply eqs. (1.11), (1.12) to a reservoir that deals with the PC task, we will have the notion of how many points in the past we can consider in order to perform successfully the nonlinear map, which is, as a matter of fact, the parity determination of the considered set of points. For this reason, the larger the memory function is when we consider more past points, the more nonlinear capacity will the system potentially have.
Hence, if we compute MC when we are dealing with PC task, we will be determining the nonlinear memory of the system (hereafter abbreviated as MC_{PC}), whereas if we deal with STM task we will be determining the linear memory of the system (from now on MC_{STM}).

1.6 Memory-nonlinearity trade-off

It has been reported that there is no apparent way to gain nonlinearity (MC_{PC}) without losing memory (MC_{STM}) in the process and vice versa. This is the so-called memory-nonlinearity trade-off, and it is apparently a universal relation that it has been demonstrated for several dynamical systems with different types of nonlinearities [15],[16].
This trade-off will be the basic problem to tackle along the entire master thesis and, as it turns out, there is no way to mitigate it completely.
However, we will introduce several strategies to deal with this trade-off, which means that we will be able to overcome its effect although we will not be able to suppress it.

The first technique or strategy that we will evaluate is the coexistence of linear and nonlinear virtual nodes, which will become a "combined reservoir", as introduced in [15]. Without changing the topology of the reservoir, we will include some nodes with the following linear activation function (which is, in fact, Eq. (1.3) without considering the nonlinear part):

$$r_i(n) = \gamma W_i^{\text{in}} x(n) + \beta r_{i-1}(n-1) \tag{1.13}$$

However, as we will see in the results chapter, this new type of nodes will improve the linear memory of the system, but, in general, it will induce a lack of nonlinearity in the system. For that reason, this is not a general solution for all tasks.

In order to complement the effect of introducing linear nodes, if we want to deal with the memory-nonlinearity trade-off, we suggest to split the reservoir in two differentiated reservoir layers, the linear and the nonlinear one (as it is sketched in Fig. 1.5).
Thus, we will have a multilayer reservoir that will provide a lot of new possibilities to modify the properties of the reservoir performance. The behavior of the reservoir will now depend on which type of layer we inject the input to, how will be the connections between reservoirs (interlayer weights) and which reservoir will be used to carry out the training stage.

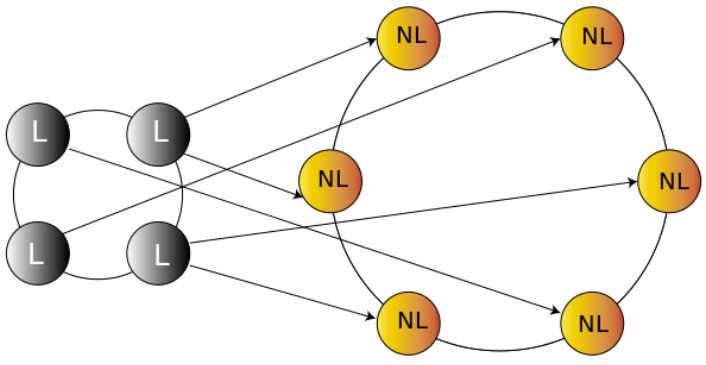

Figure 1.5: Sketch of a reservoir splitted into a linear layer and a nonlinear one.

Hence, with this global overview provided by this introductory chapter, we can now appreciate the big amount of possible scenarios that we will deal with in order to overcome the memory-nonlinearity trade-off; we will try to find out a new optimal two-layers reservoir, the optimal proportion of linear nodes and, finally, the reservoir parameters that let the multilayer reservoir perform the three tasks better than the single reservoir with only nonlinear nodes in it.

Chapter 2

Results & Calculations

During this chapter we will present all the results obtained in the process of finding out the system configuration that better compensates the memory-nonlinearity trade-off.
Since we will present multiple case studies, we will increase the complexity of the reservoir in several steps and analyze every case exhaustively.
Specifically, we will go from a single reservoir with nonlinear virtual nodes to a two-layers reservoir with interlayer connections.

2.1 Single-layer reservoir with nonlinear nodes

The first step that we have to do is to determine which will be the optimal number of virtual nodes N. To do this, we will consider Eqs. (1.11) and (1.12) in order to compute the memory capacity (MC) as a function of N. Thus, considering that N goes from 50 to 1150 in intervals of 50 and that for each N we have to introduce a new mask, the results for STM and PC tasks are the following:

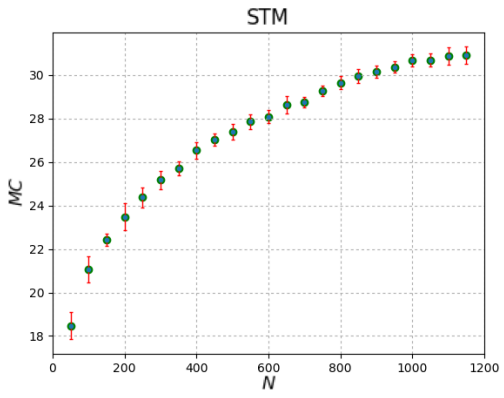

Figure 2.1: MC_{STM} as a function of the number of virtual nodes. The error bars have been determined computing the standard deviation using 10 samples for each point in the plot.

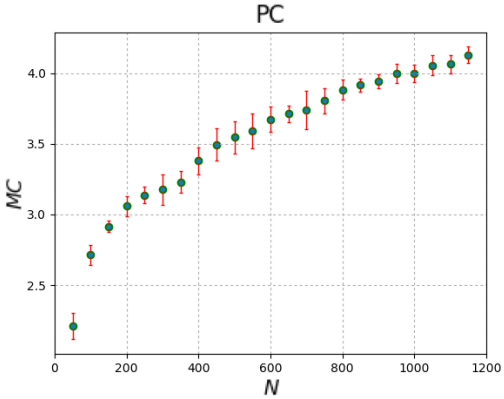

Figure 2.2: MC_{PC} as a function of the number of virtual nodes. The error bars have been determined computing the standard deviation using 10 samples for each point in the plot.

Anticipating some results, the considered parameters to make the calculations in Figs. 2.1 and 2.2 are those ones that provide us a compromise performance between STM and PC tasks, which are: $\beta = 0.8$, $\gamma = 0.1$, $\phi = 0.7\pi$.
As we can see from these figures, MC reaches a saturation regime when we increase N. For convenience, we will consider from now on reservoirs with $N = 500$, because they will achieve a good compromise between calculation speed, accurate results and the amount of necessary data for the training procedure (typically, as an empiric law, the more virtual nodes you have, the more training data you need to achieve accurate results in the testing stage).

Now that we have set the number of nodes that we will consider in all the following scenarios, it is time to find what parameter combination is the best for STM and PC tasks and verify if both tasks are compatible or not.

2.1.1 Optimization of STM task

Looking for the best parameter combination for the STM task, we will perform a parameter scan. We start setting an intermediate phase ϕ and we compute MC for several values of β and γ between $[0, 1]$, then we consider the β set with higher MC of the heat map and we perform another heat map but now varying the remaining variable ϕ. This procedure is shown in Figs. 2.3 and 2.4:

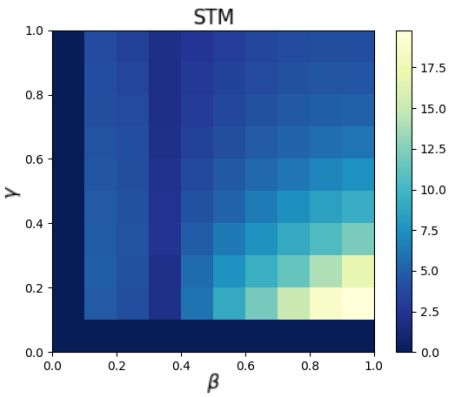

Figure 2.3: MC_{STM} heat map for the parameters β and γ, setting $\phi = 0.4\pi$.

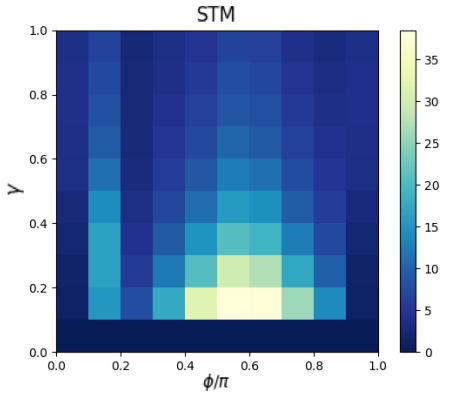

Figure 2.4: MC_{STM} heat map for γ, ϕ, setting $\beta_{opt} = 1$ as optimal β.

From Figs. 2.3 and 2.4, we can appreciate that the best parameter combination for the STM task is: $\beta = 1$, $\gamma = 0.1$ and $\phi = 0.6\pi$. The result obtained for the parameter β agrees with previous findings [5], because in this task it is required to increase as much as possible the memory of the system. For this reason, one needs to maximize the term that involves previous values in Eq. (1.3) (i.e. the term that is multiplied by parameter β) without rendering the system unstable.

If we now plot the memory function versus the past p with the set of parameters $\beta = 1$, $\gamma = 0.1$ and $\phi = 0.6\pi$, we obtain the results shown in Fig. 2.5:

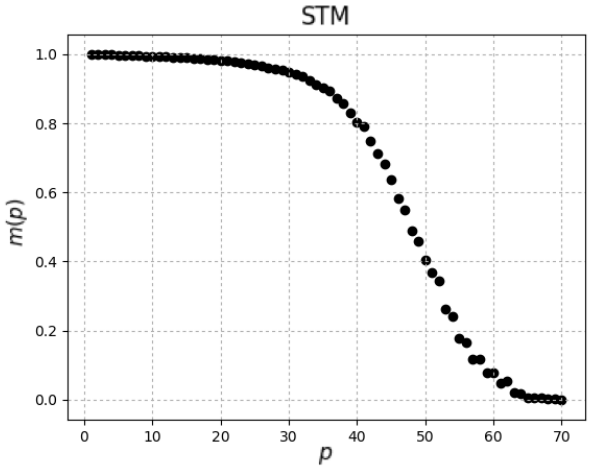

Figure 2.5: Maximum memory function considering the best parameter combination for the STM task with a ring reservoir of 500 nonlinear nodes. The result for the linear memory is $MC_{STM} = 46.55$.

2.1.2 Optimization of PC task

Now, we proceed to evaluate the parameters of the reservoir in order to achieve the best performance for the PC task. The results for the parameter scan are shown in Figs. 26 and 2.7 as heat maps:

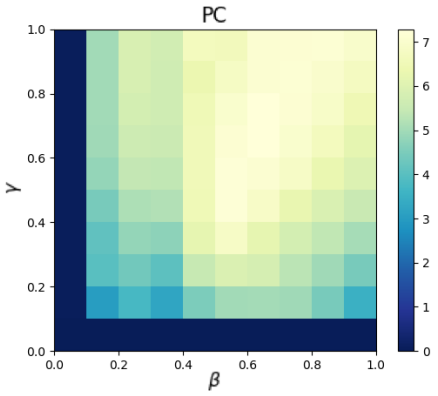

Figure 2.6: MC heat map for the parameters β and γ, setting $\phi = 0.4\pi$.

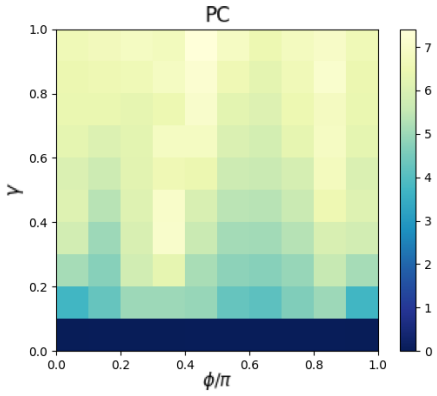

Figure 2.7: MC heat map for the parameters γ, ϕ, with $\beta_{opt} = 0.7$ as optimal β.

Looking into Figs. 2.6 and 2.7, it can be seen that the best parameter combination for the PC task is: $\beta = 0.7$, $\gamma = 1$, $\phi = 0.4\pi$.
In a similar way than in STM task, $\gamma = 1$ is an expected result because PC task requires a lot of nonlinearity, thus, we need to increase as much as possible the part that involves the nonlinear transformation of the input in Eq. (1.3) (i.e. making

relevant the term that has γ multiplying).
With the best parameter combination for this task ($\beta = 0.7$, $\gamma = 1$, $\phi = 0.4\pi$), we present the memory function versus past points in Fig. 2.8:

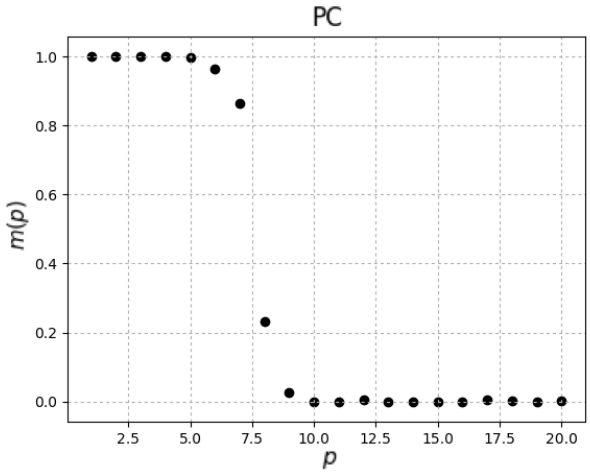

Figure 2.8: Maximum memory function considering the best parameter combination for the PC task with a ring reservoir of 500 nonlinear nodes. The result for the nonlinear memory is $MC_{PC} = 7.10$.

2.1.3 Compromise between the tasks

Once we have determined what parameters combination are the best for the STM and PC tasks, we have to try to deal with the memory-nonlinearity trade-off reaching a compromise parameters between these tasks.
Looking back at Figures 2.3 and 2.6 we can appreciate that when $\beta = 0.8$ we can reach a relatively good performance for both tasks. Thus, performing a parameter scan with this new compromise value of $\beta \equiv \beta_{\text{compr}}$, we obtain the heat maps shown in Figs. 2.9 and 2.10. From these results, we can clearly appreciate the existence of the memory-nonlinearity trade-off. At the regions on which MC_{STM} becomes relevant in Figure 2.9, MC_{PC} is very much reduced in Figure 2.10 and vice versa.

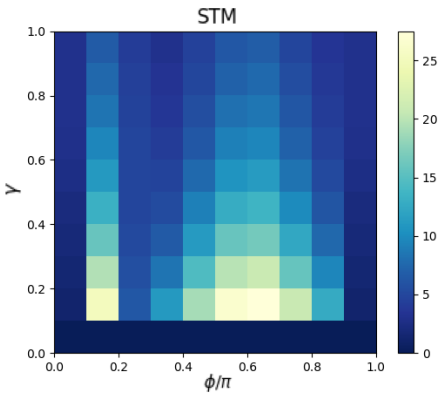

Figure 2.9: MC_{STM} heat map for the parameters γ, ϕ, with $\beta_{\text{compr}} = 0.8$.

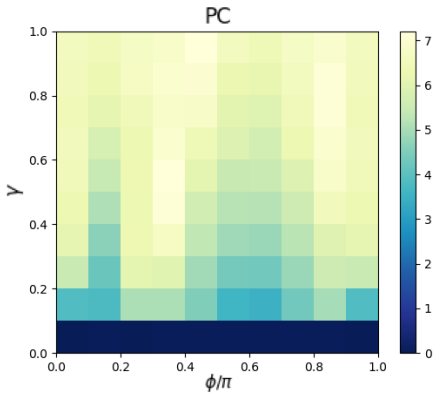

Figure 2.10: MC_{PC} heat map for the parameters γ, ϕ, with $\beta_{\text{compr}} = 0.8$.

The compromise between linear and nonlinear memory capacities, as we expected, makes impossible to reach a parameter combination that benefits both tasks simultaneously. Thus, one can set a combination of parameters for which both tasks are harmed, but not too much, considering as compromise parameters: $\beta = 0.8$, $\gamma = 0.1$, $\phi = 0.7\pi$. With this parameter combination, the performance of the reservoir for the different tasks is shown in Figs. 2.11 and 2.12:

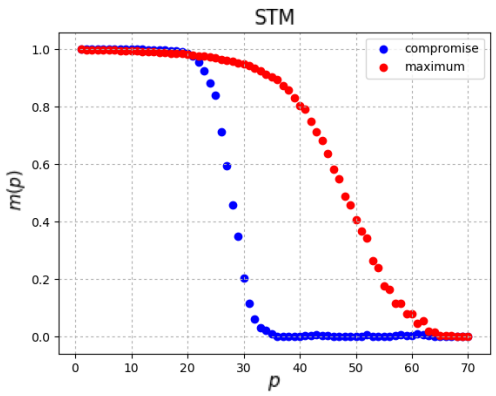

Figure 2.11: Comparison between maximum and compromise memory function versus past for the STM task.

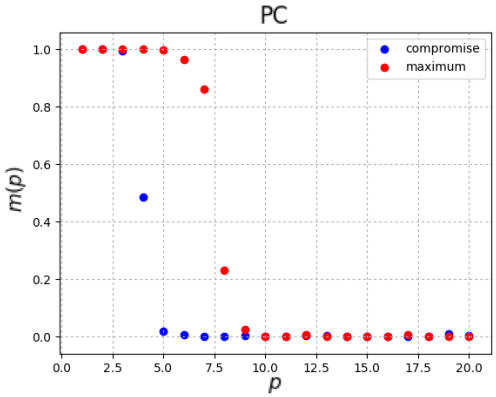

Figure 2.12: Comparison between maximum and compromise memory function versus past for the PC task.

Despite the performance getting worse for both tasks, the most harmed is the PC task because in its maximum performance it can process successfully 6 inputs in the past and with the new parameters it can only process 3 past inputs. Interestingly, with these chosen parameters, the prediction task of the MG chaotic time-series is quite accurate, as it is shown in Fig. 2.13. Hence, as the results for the three tasks are acceptable, we will use this combination of parameters ($\beta = 0.8$, $\gamma = 0.1$, $\phi = 0.7\pi$) as the baseline in the next section, in which we will introduce linear nodes in the reservoir to try to improve the performance for all tasks simultaneously.

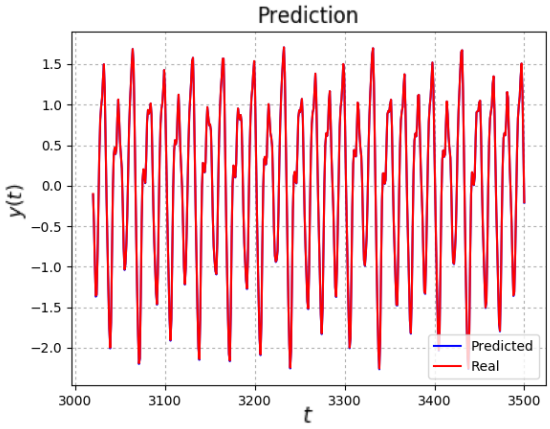

Figure 2.13: Prediction task with compromise parameters. There is no appreciable difference between the results obtained from the reservoir and the real ones, yielding a $RMSE = 9.87 \times 10^{-7}$.

2.2 Single-layer reservoir with linear & nonlinear nodes

Along this section, we will study the effects of introducing linear nodes into the reservoir following the ideas in [15]. First, we will see the consequences of putting grouped linear nodes and then we will see if there are some differences if we interleave them instead. In order to clarify the new reservoir configurations, in Fig. 2.14 we sketch a reservoir with grouped linear nodes and another one with interleaved linear nodes.

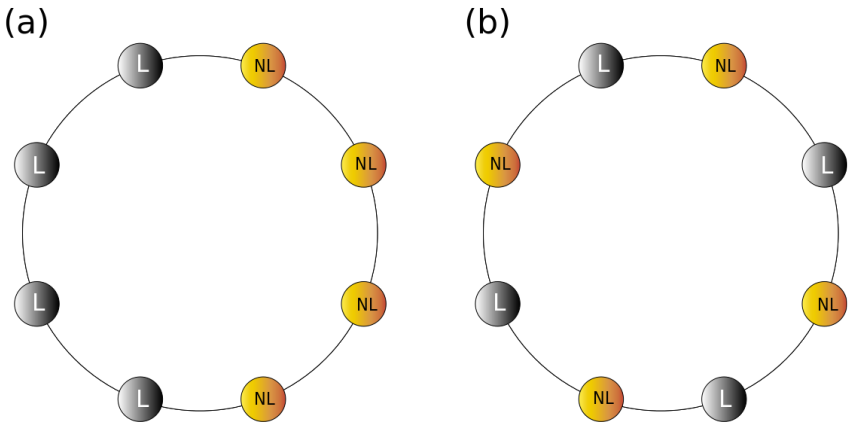

Figure 2.14: (a) Scheme of a reservoir with 50% of grouped linear nodes. (b) Scheme of a reservoir with 50% of interleaved linear nodes.

Finally, we will analyze if the new results will improve or not the previous ones coming from the single reservoir with nonlinear nodes and the compromise parameters.

2.2.1 Grouped linear and nonlinear nodes

In this section, we will put grouped linear nodes into the reservoir as it is shown in Fig. 2.14(a); selecting from the total number of nodes some percent of nodes that will follow Eq. (1.13) and considering the remaining nonlinear nodes following Eq. (1.3). Thus, we will distribute the nodes in two differentiated portions on the ring: the linear and the nonlinear part. For example, if the considered percent of linear nodes is 50% we will split the nodes ring in two equal parts, one following Eq. (1.13) (linear) and the other following Eq. (1.3) (nonlinear).

Taking this into account, Figs. 2.15 and 2.16 show the results of the previous considerations for both tasks using the compromise parameter set $\beta = 0.8$, $\gamma = 0.1$, $\phi = 0.7\pi$.

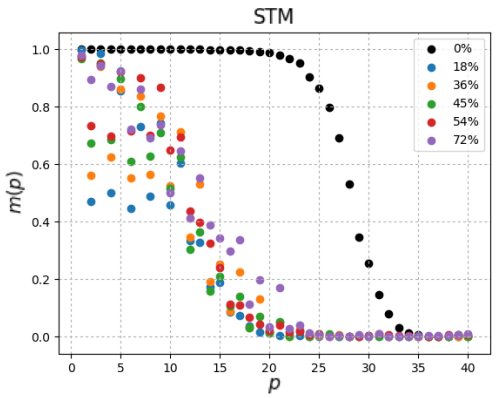

Figure 2.15: Memory function versus past for the STM task considering different percents of grouped linear nodes.

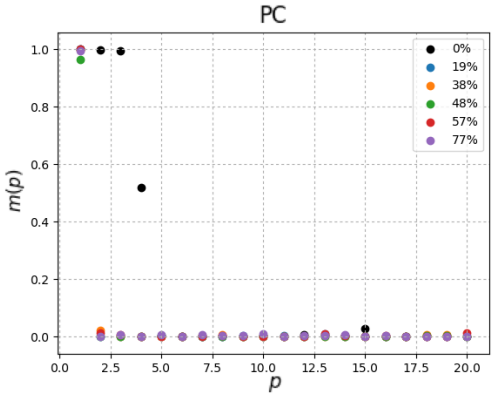

Figure 2.16: Memory function versus past for the PC task considering different percents of grouped linear nodes.

As we can see in Figs. 2.15 and 2.16, putting grouped linear nodes into the reservoir reduces drastically its performance for both the STM and the PC tasks. For this reason, we will exclude this reservoir configuration from now on.

2.2.2 Interleaved linear and nonlinear nodes

Following a similar procedure than in the previous section, we will put some linear nodes into the reservoir, but this time we will interleave them between the nonlinear nodes (see Fig. 2.14(b)).
Notice that linear nodes will be between the nonlinear ones when we will consider a percent of linear nodes below 50%. If the percent considered is above the threshold of 50%, we will have nonlinear nodes between linear nodes.

The result obtained for this scenario and for both tasks is shown in Figs. 2.17 and 2.18 for the parameter set $\beta = 0.8$, $\gamma = 0.1$, $\phi = 0.7\pi$. With this configuration, at least we have improved considerably the STM task whereas the PC task is not significantly harmed.

Beyond the third point in the past, the PC task is not performed successfully for any percent of linear nodes. Thus, we can consider that an optimal percent of linear nodes could be 80%, because, if we look at these results, the pink plot at PC task shows an accurate performance until the third point in the past, while in the STM case, the same plot increases considerably in comparison with the nonlinear reservoir (i.e. black plot with 0% label). For this configuration, $MC_{STM} = 37.86$ and $MC_{PC} = 3.02$.

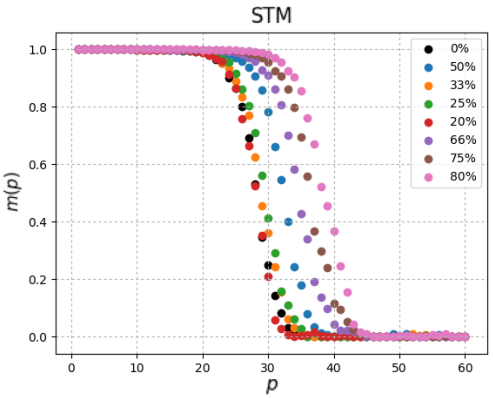

Figure 2.17: Memory function versus past for the STM task considering different percents of interleaved linear nodes.

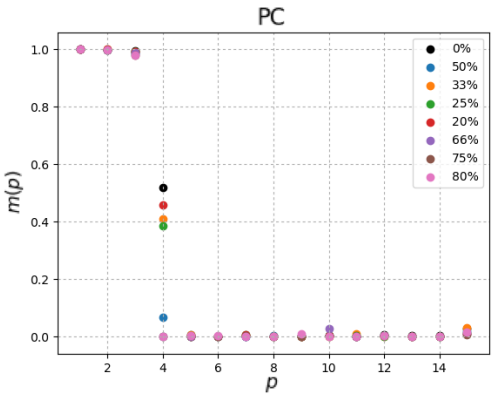

Figure 2.18: Memory function versus past for the PC task considering different percents of interleaved linear nodes.

Now that we have obtained a regime on which the STM task has improved and the PC task has not been considerably harmed, the next step is try to achieve a new parameter combination that let us improve PC task without harming the STM. First we will set the, a priori, best configuration with 80% of linear nodes and 20% of nonlinear ones, but we will also explore other combinations of these percents in order to obtain the desired regime.

In this first try, using the 80% percent of linear nodes, we perform another parameter scan as follows and show the results in Figs. 2.19 and 2.20 as heat maps. From these heat maps, we can appreciate how there is no other possibility to find another parameter combination that benefits both tasks. It can be clearly seen again the memory-nonlinearity trade-off; if we want to improve the PC task, we necessarily have to harm the STM, because PC basically requires high γ whereas STM requires high β with low γ.

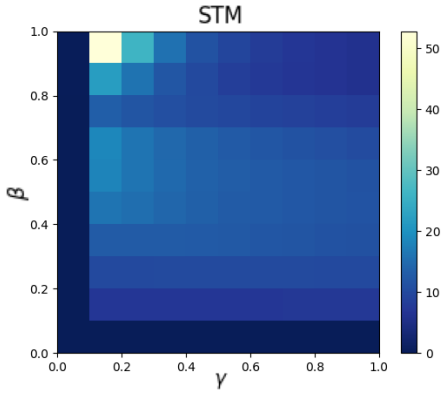

Figure 2.19: Parameter scan for β, γ, with 80% of linear nodes for the STM task, setting $\phi = 0.4\pi$.

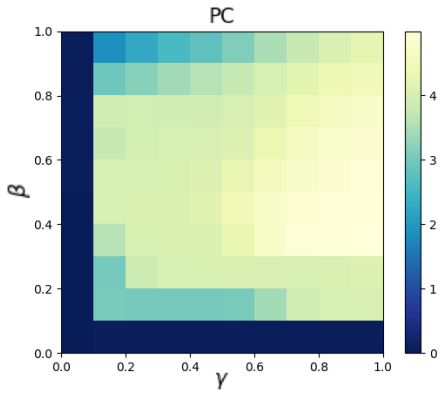

Figure 2.20: Parameter scan for β, γ, with 80% of linear nodes for the PC task, setting $\phi = 0.4\pi$.

If we now invert the percentages (i.e. setting 20% of linear nodes and 80% of nonlinear), we obtain the results sown in Figs. 2.21 and 2.22 by scanning the same parameters than before. From these heat maps, we can see that, although in Figure 2.21 the region of good performance of the STM task has been expanded with respect to Fig. 2.19, we can not reach other parameter combination because the tasks are still clearly incompatible.

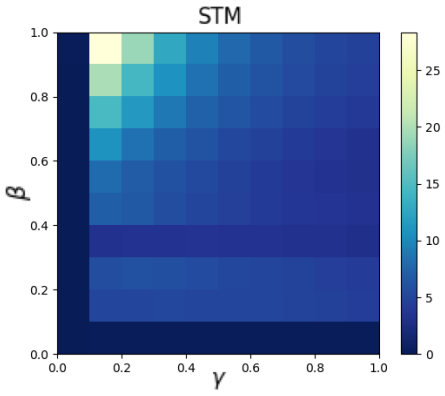

Figure 2.21: Parameter scan for β, γ, with 20% of linear nodes for the STM task, setting $\phi = 0.4\pi$.

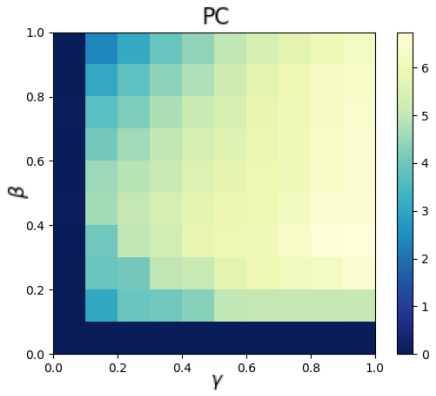

Figure 2.22: Parameter scan for β, γ, with 20% of linear nodes for the PC task, setting $\phi = 0.4\pi$.

Finally, considering 50% of linear and nonlinear nodes, we again perform a parameter scan to try to find the regime on which PC is improved without harming STM. In this case, we can see in Figs. 2.23 and 2.24 how there is, perhaps, an option to improve the PC task harming as little as possible the STM task, and this is considering $\beta = 0.6$. Thus, we again do a parameter scan for γ and ϕ setting $\beta = 0.6$. From the heat maps in figures 2.25 and 2.26, we can see how there is no combination parameters for which the PC task is considerably improved while the STM is not considerably harmed. Moreover, with this percent of linear nodes we can only obtain, in the best case, a good performance below 20 past points in the STM task.

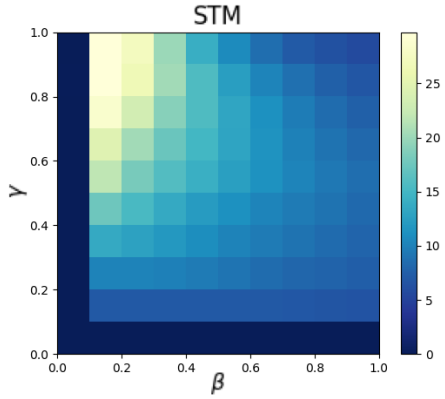

Figure 2.23: Parameter scan for β, γ, with 50% of linear nodes for the STM task, setting $\phi = 0.4\pi$.

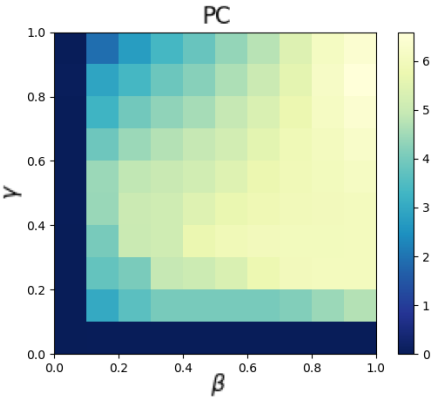

Figure 2.24: Parameter scan for β, γ, with 50% of linear nodes for the PC task, setting $\phi = 0.4\pi$.

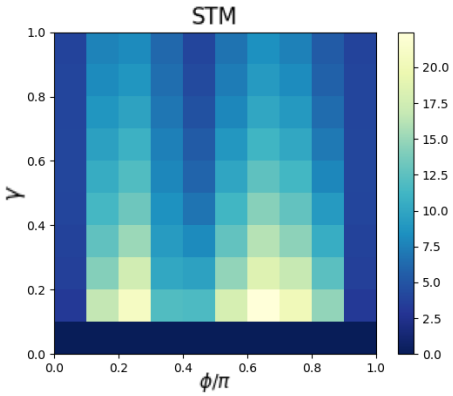

Figure 2.25: Parameter scan for γ, ϕ, with 50% of linear nodes for the STM task, setting $\beta = 0.6$.

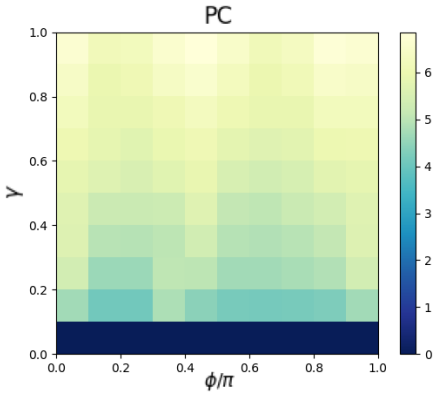

Figure 2.26: Parameter scan for γ, ϕ, with 50% of linear nodes for the PC task, setting $\beta = 0.6$.

Using a reservoir with interleaved linear and nonlinear nodes, we have obtained a result that improves considerably the STM task but leaves the PC task with a relatively low performance, even exploring the percentage of linear nodes and other parameter sets. If one wants the reservoir to obtain simultaneously a good result for both tasks, we will have to explore other configurations as shown in the following section.

2.3 Multilayer reservoirs

Here, we will develop the main contribution of this master thesis, that is splitting the reservoir into several rings, one nonlinear and the other one linear (see Fig. 2.27).
Along this section we will study the consequences of introducing this feature in the reservoir, but before that, we have to explain the basic concepts that are related with this new scheme.

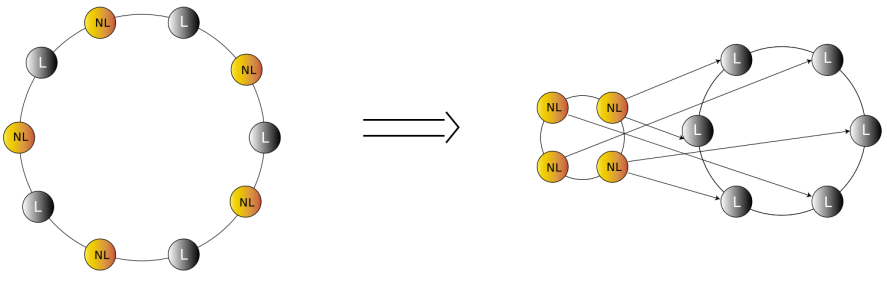

Figure 2.27: Sketch of the splitting process of the reservoir with interleaved linear and nonlinear nodes. Notice that now we have interlayer weights (which are represented by the arrows between the reservoir layers) and that in this new configuration the input could be injected to both reservoir layers.

The first thing that we will have to take into account is at which reservoir will the input go. As we will see, it will not be the same introducing the input data into the nonlinear reservoir than in the linear one.

Another important thing to consider are the connections between the reservoirs or interlayer weights, which will always be a random gaussian number between $[-1, 1]$. There are many possibilities to connect both reservoirs since we will not necessarily deal with linear and nonlinear reservoirs with the same number of nodes.

One of the considered combinations will be random connections (illustrated in Fig. 2.28(c)); each node of the biggest reservoir will be connected randomly with one node of the smallest one. The connection probability of the node of the big reservoir to the small reservoir will be uniform (i.e. each node of the smaller reservoir will have the same probability to be connected with one node of the bigger one).

Another possibility will be to consider also non-random connections between reservoirs. This will consist in connecting proportionally all the nodes of the smaller reservoir with the nodes of the big one. For example, if we have a linear reservoir with 100 nodes and a nonlinear one with 400 nodes (i.e. if we have split a reservoir with 80% of nonlinear interleaved nodes and 20% of linear interleaved nodes), we will connect the first node of the smaller reservoir with the first four ones of the bigger reservoir, the second node of the small reservoir with the nodes from the fifth to the eighth of the big one, and so on with the rest of nodes.
Moreover, following the previous example, we find no appreciable difference between considering the same intermediate weight in the nodes connection 1 to 4, than considering different weights in each connection. In other words, the result is the same considering a unique w_i as intermediate weight between the node of the smaller reservoir and its correspondent nodes of the big one (see Fig. 2.28(a)), than considering a set of weights $\{w_i, w_j, w_k, w_l\}$ to connect the nodes in question (see Fig. 2.28(b)). Thus, for simplicity, we will consider, in most of the cases, the scheme on which we only have one weight w_i for every nodes connection 1 to 4.

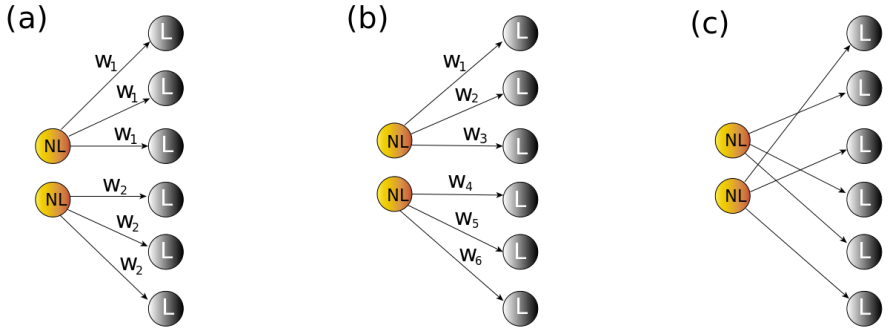

Figure 2.28: (a) Sketch of the 1 to 3 interlayer nodes connections using the same intermediate weight w_i. (b) Sketch of the 1 to 3 interlayer nodes connections using a set of weights $\{w_i, w_j, w_k\}$. (c) Sketch of interlayer connections selecting nodes randomly.

Finally, the last key point in two-layer reservoirs will be which reservoir will be used in the training process. Along the section results, we will specify if we have considered both layers or only one of them to determine the output weights.
Now that we have explained the main characteristics of this new scenario, we will present the obtained results in the following subsections.

2.3.1 Preliminary study of the two-layer reservoir

Initially, we will consider the compromise parameters defined in section 2.1.3 (i.e. $\beta = 0.8$, $\gamma = 0.1$, $\phi = 0.7\pi$) in the case of a combined reservoir with 80% of linear nodes and 20% of nonlinear. This time, however, we split the reservoir into a ring layer with 100 nonlinear nodes and another one with 400 linear nodes.
Once we have set the reservoir configuration, we first show in Fig. 2.29 that there is no quantitative difference between considering a unique intermediate weight w_i than considering a set of different weights $\{w_i, w_j, w_k, w_l\}$ for the nodes connections 1 to 4. We have chosen the STM task to illustrate this finding for simplicity.

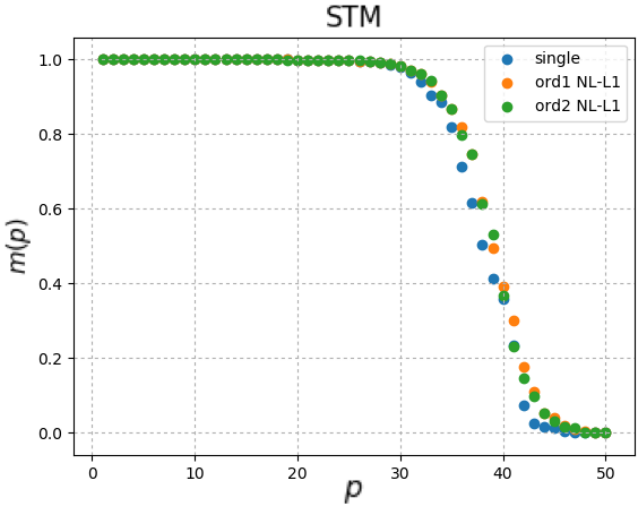

Figure 2.29: Memory function versus past for the STM. At the legend, "Ord1" and "Ord2" represent the intermediate weights w_i and $\{w_i, w_j, w_k, w_l\}$, respectively. NL-L1 indicates that the input has been introduced in the nonlinear reservoir and the training process has been performed with both reservoir layers.

We can start noticing that the results for the two-layer reservoir are slightly better than for a single reservoir. Furthermore we can see how there is no significant difference between the two alternative interlayer connectivity cases mentioned earlier.

In the following, we will use a simplified notation in the plot legends. As we have seen, "ord1" and "ord2" represent using either w_i or $\{w_i, w_j, w_k, w_l\}$ as intermediate weights, respectively. In turn, "rand" refers that the nodes of the different layers are randomly connected. And finally, we will use "NL-Ln" if the input is introduced into the nonlinear reservoir and "L-NLn" if it is introduced into the linear one, whereas $n = 1, 2$ means that the training process is performed with

either both reservoir layers or with a single reservoir layer that has not been used to introduce the input, respectively.
Once we have set the notation, we present in Figs. 2.30 and 2.31 the results for both tasks considering that the input data goes to the nonlinear reservoir. Although it does not represent a relevant difference, the case of NL-L1 (i.e. the orange plot in figures 2.30 and 2.31) is remarkable since both tasks are simultaneously improved. This improvement is barely appreciable, but it represents a good indication that two-layer reservoirs are the right way in order to deal with the memory-nonlinearity trade-off.

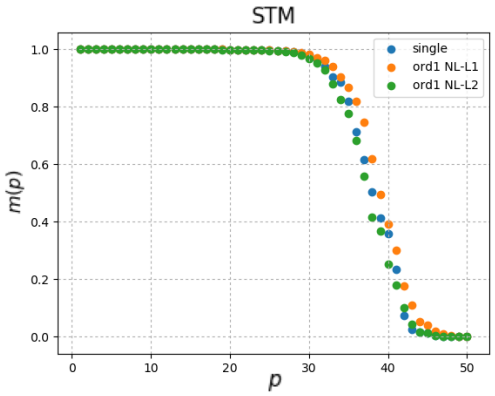

Figure 2.30: Memory function versus past for different schemes for the STM task.

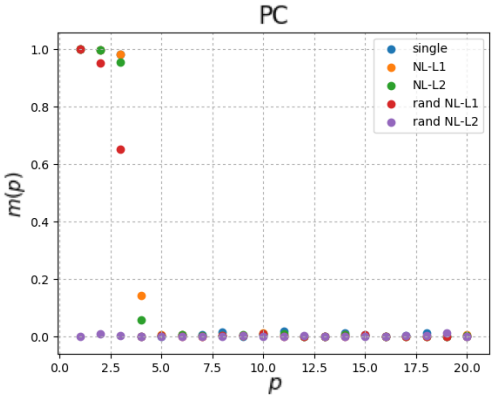

Figure 2.31: Memory function versus past for different schemes for the PC task.

For completeness, we show that, when we use the nonlinear reservoir to introduce the input data, random intermediate connections are useless and only harm the reservoir performance. In Fig. 2.31, if we look at red and purple plots, we can already observe this fact in the PC task. Fig. 2.32 shows the same behavior but for the STM task.

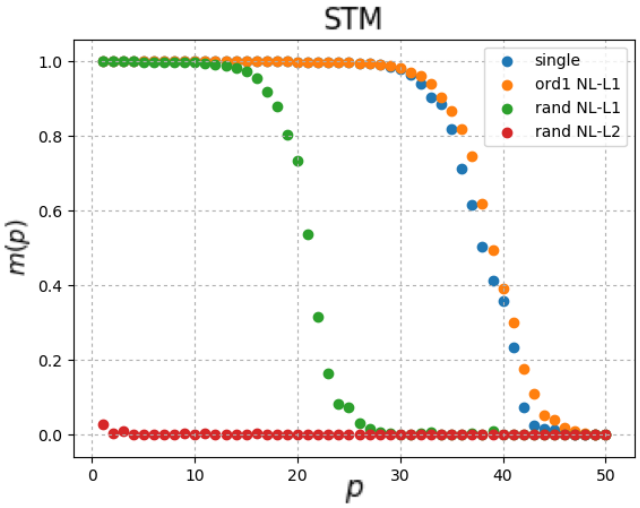

Figure 2.32: Memory function versus past for different schemes emphasizing the bad performance of the reservoir having random interlayer connections in the STM task.

If we now invert the order of the reservoirs and we introduce the input into the linear reservoir, we obtain the results shown in Figs. 2.33 and 2.34. First of all, in Figure 2.33 we can see how there is no difference between the purple and green plot (i.e. the cases L-NL1 with random connections and proportional connections provide exactly the same result). Thus, for simplicity, we will consider the case "L-NL1" as the best one for the STM task, disregarding random connections because they imply harder experimental implementations.

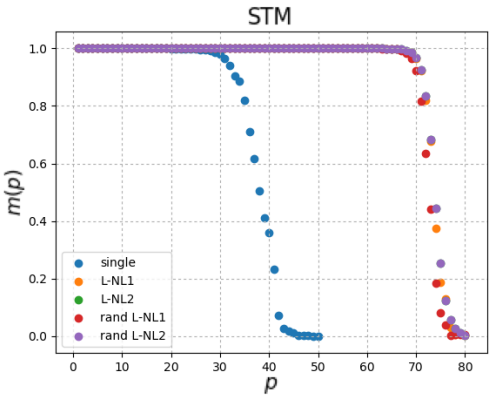

Figure 2.33: Memory function versus past for different schemes for the STM task.

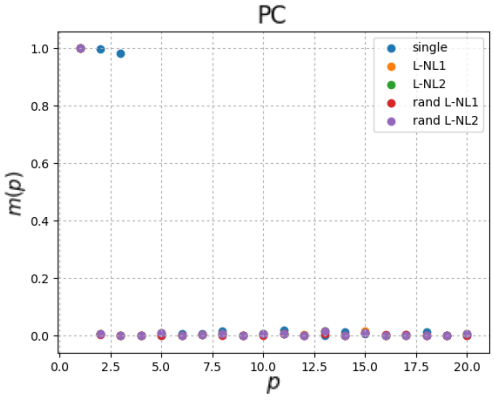

Figure 2.34: Memory function versus past for different schemes for the PC task.

Fig. 2.34 shows that the PC task is completely lost when we introduce the input into the linear reservoir, while the STM task improves considerably in Fig. 2.33 (actually the maximum of the memory function is extended practically twice). Thus, we will use this result as a hint for the next section in order to establish an ansatz that improves the current performances of the two-layer reservoir.

2.3.2 Optimization of the two-layer reservoir

By now, we know that, if we want a good performance for the PC task, we need a high degree of nonlinearity. This finding implies that if we have too many linear nodes in one of the reservoir layers the memory function of this task will be harmed, unless we introduce the input into the nonlinear reservoir.
In contrast, the STM task does not improve considerably when we put the input into the nonlinear reservoir, but it really does when we introduce it into the linear one.

For these reasons, we propose an ansatz that consists in having a nonlinear reservoir layer of 400 nodes and a linear one of 100 nodes (i.e. we split a combined reservoir with 80% of nonlinear nodes and 20% of linear), in order to improve the performance of the PC task. It has to be said that, from now on, we will determine the output weights using both reservoir layers.

With this new configuration shown in Fig. 2.35, we will have much more nonlinear nodes, which will necessarily provide nonlinearity, but, furthermore, we will introduce the input into the linear reservoir in order to try to improve simultaneously the STM task too.

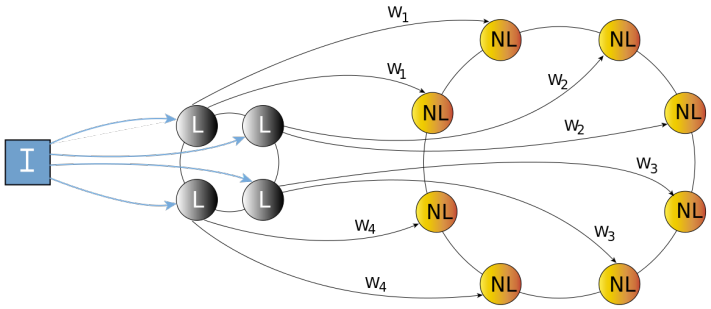

Figure 2.35: Sketch of the ansatz configuration injecting the input to the linear reservoir layer. Notice that the connections between the reservoir layers are represented as 1 to 2 and using the same interlayer weight w_i. In our simulations, our interlayer connections will be 1 to 4 nodes.

Hence, with this new configuration, and knowing that the performance of the PC task is more sensitive, we perform an additional parameter scan in order to find the new optimal parameter combination for the PC task.
The corresponding results are shown in Figs. 2.36 and 2.37.

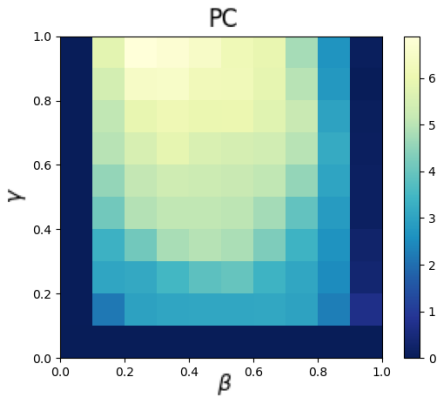

Figure 2.36: Heat map of γ, β, for the PC task, setting $\phi = 0.4\pi$.

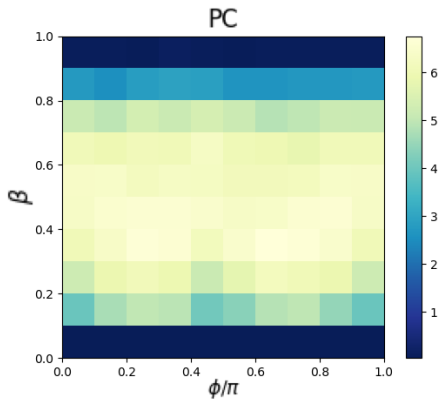

Figure 2.37: Heat map of β, ϕ, for the PC task, setting $\gamma_{opt} = 1$.

In a first attempt, we consider from the previous figures the best parameter combination ($\beta = 0.3$, $\gamma = 1$, $\phi = 0.7\pi$) in order to try to maximize the performance for the PC task without disregarding the STM task.

The results of the configuration shown in Fig. 2.35 with the optimal parameters for the PC task are shown in figures 2.38 and 2.39, and they are still quite restrictive for the STM task.

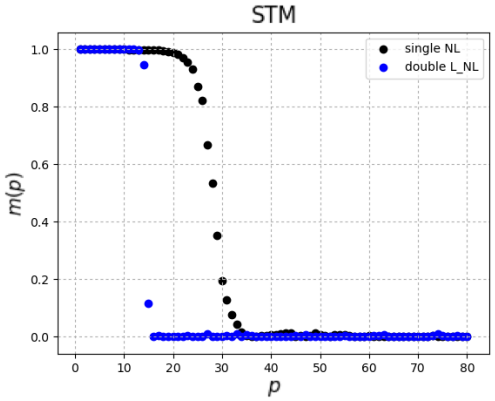

Figure 2.38: Memory function versus past, for the STM task. Comparison between the new configuration (with optimal parameters for the PC task) and the compromise case of single layer reservoir with nonlinear nodes.

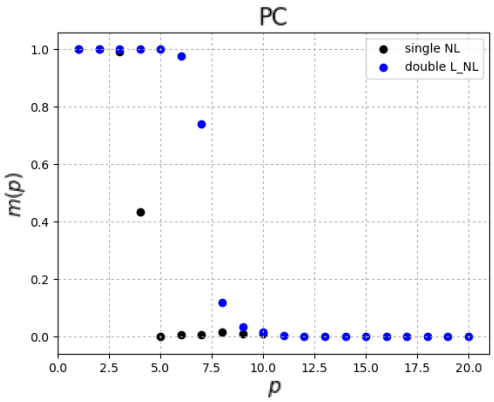

Figure 2.39: Memory function versus past, for the PC task. Comparison between the new configuration (with optimal parameters for the PC task) and the compromise case of single layer reservoir with nonlinear nodes.

In Figure 2.39 we can see that, with this new configuration, the performance for the PC task is twice better than the one obtained in section 2.1 (i.e. the compromise case of a fully nonlinear reservoir), whereas for the STM task we can see that its performance is considerably reduced.

For this reason, we look again at figures 2.36 and 2.37 in order to find a parameter combination that clearly benefits to the PC task, but without disregarding completely the STM task. Now, we will not show a parameter scan for the STM task since we already know from section 2.1 that the STM task needs a larger value of β. Hence, we have increased β until we have found the situation where both the STM and the PC tasks are considerably improved.

Considering the combination: $\beta = 0.66$, $\gamma = 1$ and $\phi = 0.44\pi$, the results for both tasks are shown in Figs. 2.40 and 2.41. From these figures we can see how we have finally achieved a configuration with a unique parameter combination that improves simultaneously the performance of the system for both tasks compared with the compromise case determined in section 2.1. The memory capacities are now $MC_{STM} = 41.42$ and $MC_{PC} = 6.35$, respectively.

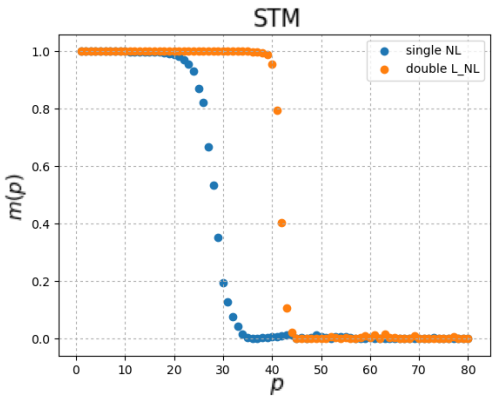

Figure 2.40: Memory function versus past, for the STM task. Comparison between the new configuration (with optimal parameters for both tasks) and the compromise case of nonlinear reservoirs.

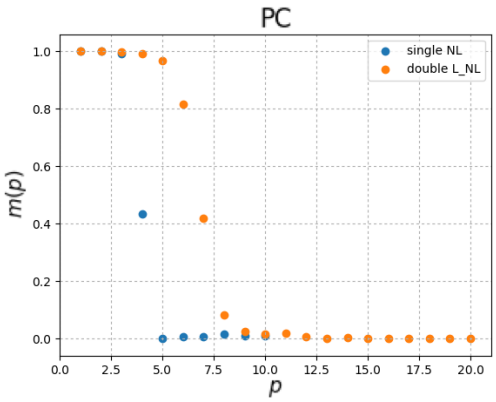

Figure 2.41: Memory function versus past, for the PC task. Comparison between the new configuration (with optimal parameters for both tasks) and the compromise case of nonlinear reservoirs.

Furthermore, as we have a good performance for both tasks, we can expect also better results for the prediction task of the MG chaotic time series. The results in question are shown in Fig. 2.42.

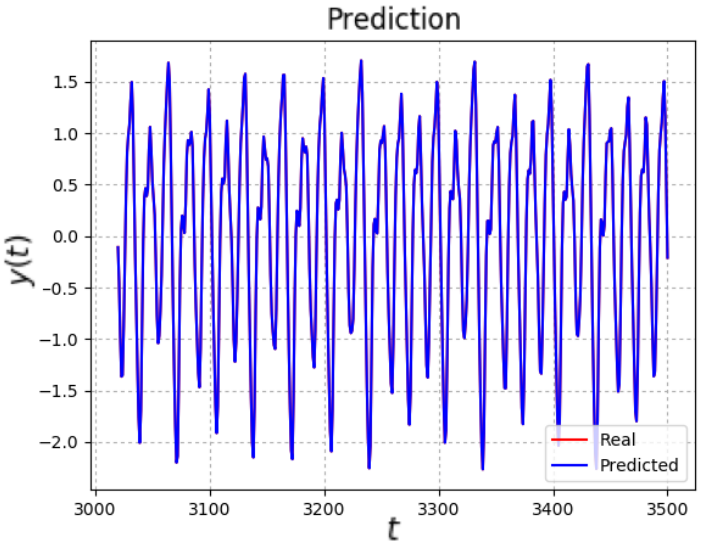

Figure 2.42: Prediction task with the definitive configuration of the two-layers reservoir with the optimal parameters ($\beta = 0.66$, $\gamma = 1$ and $\phi = 0.44\pi$).

As we guessed, again we have an exact match between the predicted results from the reservoir and the real ones for the prediction task. However, this time we have a $RMSE = 6.01 \times 10^{-7}$, which is, as a matter of fact, a lower error than the obtained in Fig. 2.13 (which is $RMSE = 9.87 \times 10^{-7}$). Thus, we have verified the improvement of the system performance for the three tasks in question and for an optimized reservoir topology.

Chapter 3

Final discussion & Conclusions

From our numerical results, we can extract several relevant conclusions and properties of reservoir computing.
Firstly, we now know that the precise distribution of linear nodes in reservoir computers is directly related with the performance of the system. In section 2.2.1, we have explicitly checked that grouping the linear nodes destroys completely the performance of the reservoir. This fact takes one step further, it is not only relevant the composition of the reservoir but it is also important how the nodes are interconnected between them. As we have seen in section 2.3.2, although we have the linear nodes isolated in a linear reservoir layer, each of them are connected with the nonlinear nodes from the other reservoir layer and we have achieved the best performance for both tasks simultaneously. Thus, connections between the different types of nodes are essential.
Second, we have finally reached in section 2.3.2 the regime on which the memory-nonlinearity trade-off is compensated to a maximum degree. We illustrate the relevance of the obtained results in Figs. 3.1 and 3.2.

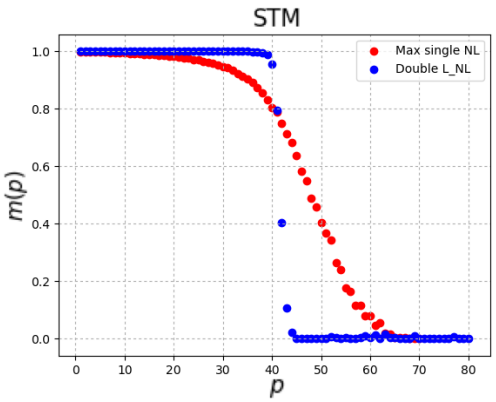

Figure 3.1: Memory function versus past, for the STM task. Comparison between the best configuration of the two-layers reservoir ($\beta = 0.66$, $\gamma = 1$ and $\phi = 0.44\pi$) and the best case of single reservoir layer with nonlinear nodes for the STM task ($\beta = 1$, $\gamma = 0.1$ and $\phi = 0.6\pi$).

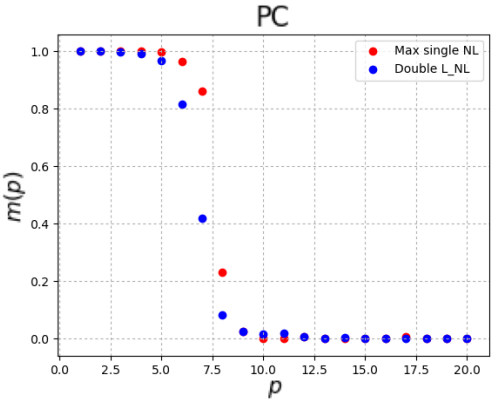

Figure 3.2: Memory function versus past, for the PC task. Comparison between the best configuration of the two-layers reservoir ($\beta = 0.66$, $\gamma = 1$ and $\phi = 0.44\pi$) and the best case of single reservoir layer with nonlinear nodes for the PC task ($\beta = 0.7$, $\gamma = 1$ and $\phi = 0.4\pi$).

From the blue plots of the previous figures, we can observe the goodness of the new configuration. With the same parameters and configuration (i.e. a linear reservoir layer, that receives the input, connected proportionally with the nonlinear one and with a unique parameter set $\{\beta, \gamma, \phi\} = \{0.66, 1, 0.44\pi\}$), we practically reproduce the best performance of a single reservoir layer with nonlinear nodes with the best parameter combination for the PC task and, furthermore, we improve the results coming from the single nonlinear reservoir with the best parameter combination for the STM task.
In this last case, in Figure 3.1, we can see how we do not have the same fading out memory effect than in the fully nonlinear case (i.e. red plot), but we obtain perfect results for further past points. That is the reason why we say that the results are better with this new configuration.

The results in this master thesis yield an improvement over the current techniques of reservoir computing because of its transversality. With only one configuration and one parameter set, the system is prepared to deal with the so-called memory-nonlinearity trade-off and it can perform successfully different tasks that, a priori, are opposite because of such a trade-off.

We anticipate that this study could be further expanded by exploring different scenarios and applications in order to verify if this kind of new configurations remain generic for other tasks, or if they could be even more optimized.

Bibliography

[1] L. Appeltant. 'Reservoir computing based on delay-dynamical systems'. (PhD thesis), *Virje Universiteit Brussel* VUB, *Instituto de Física Interdisciplinar y de Sistemas Complejos* IFISC-UIB (2012).

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

[3] M. Maass, T. Natschläger, H. Markram. 'Real-time computing without stable states: A new framework for neural computation based on perturbations'. *Neural Computation*, 14(11):2531–2560, (2002).

[4] D. Brunner, M. C. Soriano, G. Van der Sande. *Photonic Reservoir Computing, Optical Recurrent Neural Networks*, Chapter 5, De Gruyter (2019).

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

[6] M. Lukoševičius. "A practical guide to applying echo state networks". *Neural networks: Tricks of the trade*, 659-686, DOI:10.1007/978-3-642-35289-8_36, (2012).

[7] L. Appeltant, M.C. Soriano, G. Van der Sande1, J. Danckaert, S. Massar, J. Dambre, B. Schrauwen, C.R. Mirasso & I. Fischer. 'Information processing using a single dynamical node as complex system', *Nature Communications*, 2:468, DOI: 10.1038/ncomms1476 (2011).

[8] J. Bueno, D. Brunner, M.C. Soriano & I. Fischer. "Conditions for reservoir computing performance using semiconductor lasers with delayed optical feedback". *Optics express* 25 (3), 2401-2412 (2017)

[9] H. Jaeger. 'The "echo state" approach to analysing and training recurrent neural networks – with an Erratum note'. *German National Research Institute for Computer Science*, (2010).

[10] T. Cover, 'Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition'. *IEEE Trans. Electron.* 14 , p. 326–334, (1965).

[11] S. Ortín, M. C. Soriano, L. Pesquera, D. Brunner, D. San-Martín, I. Fischer, C. R. Mirasso & J. M. Gutiérrez. 'A Unified Framework for Reservoir Computing and Extreme Learning Machines based on a Single Time-delayed Neuron'. *Scientific Reports*, 5:14945, DOI: 10.1038/srep14945

[12] A. Rodan & P. Tiňo. 'Simple deterministically constructed cycle reservoirs with regular jumps'. *Neural computation*, 24, 1822–1852 (2012).

[13] M. C. Mackey & L. Glass. 'Oscilation and Chaos in Physiological Control Systems'. *Science*, New Series, Volume 197 (1977)

[14] K. Fujii & K. Nakajima. 'Harnessing Disordered-Ensemble Quantum Dynamics for Machine Learning'. *Physical Review Applied*, 8, 024030 (2017)

[15] M. Inubushi & K. Yoshimura. 'Reservoir computing beyond memory-nonlinearity trade-off'. *Scientific reports*, 7:10199, DOI:10.1038/s41598-017-10257-6 (2017)

[16] J. Dambre, D. Verstraeten, B. Schrauwen & S. Massar. "Information Processing Capacity of Dynamical Systems". *Scientific Reports* 2, 514, doi: 10.1038/srep00514 (2012).