Aristotle University of Thessaloniki
Faculty of Sciences
School of Physics

Reservoir Computing in chaotic time series forecasting: A study of the influence of reservoir structure on the quality of predictions

A thesis submitted in partial fulfillment for the
Undergraduate Degree

by

Baresis Giannis-15134

Supervising Professor: Dr. Antoniades P. Ioannis

Thessaloniki, October 2023

Examining Committee:

1. *Dr. Ioannis Antoniades (Supervisor)*
   Special Teaching Staff
   Department of Physics, Aristotle University of Thessaloniki
2. *Dr. Ioannis Stouboulos*
   Professor
   Department of Physics, Aristotle University of Thessaloniki
3. *Dr. Konstantina Kiritsi*
   Special Teaching Staff
   Department of Physics, Aristotle University of Thessaloniki

Date of Examination: October 17, 2023

"I hereby solemnly affirm and declare that this thesis has been composed solely by myself and that it has not been submitted, in whole or in part, in any previous application for a degree. Except where states otherwise by reference or acknowledgment, the work presented is entirely my own."

Abstract

The present Bachelor's Thesis conducts an extensive investigation of the performance of Echo State Networks (ESN), a particular type of Reservoir Computer (RC), on a chaotic time-series forecasting task. The primary objective is to deepen our understanding on how to optimize an ESN type RC for complex time-series forecasting problems. Specifically, the focus was given on hyper-parameter optimization and on selecting the optimal structure and inter-connectivity of the hidden reservoir layer. The Vosvdra non-linear system of ODEs, which describes a macro-economic model of three independent variables, was used to generate the chaotic time-series. The study utilized an already developed but yet unpublished ESN model (MultiESN) which possessed great versatility in prescribing the structure of the Reservoir layer as well as how the input signal is fed into it and how its output is extracted. Thus, the outcomes of this study yield invaluable insights into how to configure ESN hyper-parameters and especially Reservoir layer structure to ensure optimal performance with the least ESN size and number of interconnections. The latter is of critical importance in reducing training duration and the cost of possible physical implementations. The ESN was evaluated both on average performance -calculated over an extended time period- but also in terms of the robustness of its forecasting ability per each time-series point. It was concluded that in order to decide on the size, geometry and hyper-parameter configuration of the ESN it is important to consider the robustness of predictions, i.e. how uniform the ESN forecasting ability is over time, not merely on average. Furthermore, this research highlights the vast potential of Reservoir Computing within the domains of Machine Learning and Artificial Intelligence, as well as its promise for future physical implementations, even though that is not the focus of this thesis. The findings provide a solid foundation for future research initiatives and practical implementation within these dynamic fields. As the digital landscape evolves, the promise of Reservoir Computing stands as a testament to the ongoing advancement of cutting-edge technologies, promising new horizons in data analysis and predictive modeling.

Εισαγωγή

Η παρούσα πτυχιακή εργασία διεξάγει μια εκτενή έρευνα σχετικά με την απόδοση των Echo State Networks (ESN), ένα ειδικό είδος του Υπολογιστή Δεξαμενής (RC), σε ένα πρόβλημα πρόβλεψης χρονοσειράς με χαοτική συμπεριφορά. Ο κύριος στόχος είναι να ενισχύσουμε την κατανόησή μας σχετικά με το πώς να βελτιστοποιήσουμε έναν τύπου ESN υπολογιστή δεξαμενής για πολύπλοκα προβλήματα πρόβλεψης χρονοσειρών. Πιο συγκεκριμένα, επικεντρωνόμαστε στη βελτιστοποίηση των υπερ-παραμέτρων και στην επιλογή της βέλτιστης δομής και των αμοιβαίων συνδέσεων μεταξύ των κρυμμένων επιπέδων. Για τη δημιουργία της χαοτικής χρονοσειράς, χρησιμοποιήθηκε το $Vosvdra$ μη γραμμικό σύστημα $ODEs$, που περιγράφει ένα μακροοικονομικό μοντέλο με τρεις ανεξάρτητες μεταβλητές. Η μελέτη χρησιμοποίησε το ήδη αναπτυγμένο, αλλά όχι ακόμη δημοσιευμένο λογισμικό $MultiESN$. Το συγκεκριμένο λογισμικό παρέχει μεγάλη ευελιξία στην δημιουργία της δομής του επιπέδου της δεξαμενής, στο πώς η είσοδος σήματος διατίθεται σε αυτό, αλλά και στο πώς παράγεται το σήμα της εξόδου. Έτσι, τα αποτελέσματα αυτής της μελέτης προσφέρουν ανεκτίμητα συμπεράσματα για το πώς πρέπει να ρυθμιστούν οι υπερ-παράμετροι του ESN και πιο αναλυτικά, για τη δομή του RC έτσι ώστε να εξασφαλιστεί η βέλτιστη απόδοση με το μικρότερο δυνατό ESN σε συνδυασμό με τον ελάχιστο αριθμό των αμοιβαίων συνδέσεων. Το τελευταίο είναι κρίσιμης σημασίας τόσο για τη μείωση της διάρκειας εκπαίδευσης, όσο και του κόστους πιθανών φυσικών υλοποιήσεων. Το ESN αξιολογήθηκε τόσο σε θέματα μέσης απόδοσης - υπολογισμένα κατά μήκος εκτεταμένης χρονικής περιόδου - όσο και σε θέματα ευρωστίας στην προβλεπτική του ικανότητα για κάθε σημείο της χρονοσειράς. Συμπεράναμε ότι για να αποφασίσουμε για το μέγεθος, τη γεωμετρία και τη ρύθμιση των υπερ-παραμέτρων του ESN, είναι σημαντικό να ληφθεί υπόψη η ευρωστία των προβλέψεων, δηλαδή πόσο ομοιόμορφη είναι η προβλεπτική ικανότητα του ESN μέσα στο χρόνο και όχι μόνο κατά μέσο όρο. Επιπλέον, αυτή η έρευνα υπογραμμίζει τη μεγάλη δυναμική του Υπολογιστή Δεξαμενής στους τομείς της Μηχανικής Μάθησης και της Τεχνητής Νοημοσύνης, καθώς και την υπόσχεσή της για μελλοντικές φυσικές υλοποιήσεις, αν και αυτό δεν αποτελεί αυτοσκοπό της διατριβής. Τα ευρήματα παρέχουν μια στέρεα βάση για μελλοντικές πρωτοβουλίες έρευνας και πρακτική υλοποίηση σε αυτούς τους ραγδαία αναπτυσσόμενους τομείς. Καθώς ο ψηφιακός κόσμος εξελίσσεται, ο Υπολογιστής Δεξαμενής στέκεται ως απόδειξη της συνεχούς προόδου των προηγμένων τεχνολογιών, υποσχόμενος νέους ορίζοντες στην ανάλυση δεδομένων και την προγνωστική μοντελοποίηση.

Acknowledgements

I am forever grateful to my supervisor, Dr. *Ioannis Antoniades*, for all the valuable advice during that year. His expertise and encouragement helped me to complete this research and write this dissertation.

I would also like to thank Dr. *Christos Volos*, the director of the Department of Applied Physics and Environmental Physics, for his significant help and contribution, as well as the Laboratory of Nonlinear Systems-Circuits and Complexity *(LaNSCom)*.

Finally, I would like to acknowledge the support provided by the IT Center of the Aristotle University of Thessaloniki (AUTh) throughout the progress of this research work, as most of the results presented have been produced using the AUTh High Performance Computing Infrastructure and Resources *(Aristotelis Docs)*.

Contents

Chapter 1

General Introduction to Reservoir Computing

Recurrent Neural Networks (RNNs) are notoriously challenging to train, leading to a general reluctance among many users to employ them in their applications. RNNs, on the other hand, serve as an immensely potent and versatile tool, combining extensive dynamic memory and adaptable computational abilities. They closely resemble biological brains, resembling the foundation of natural intelligence in Machine Learning (ML) models. Both Biological and Artificial Neural Networks (ANNs) send and receive feedback signals through their connections which partially determines their expressed state. Therefore these networks are constantly updating with information from external sources as well as internally within the network from other neurons.

Reservoir Computing (RC) is a broad subject that combines the fields of ANNs and complex dynamical systems. That interesting framework has emerged a few years ago directly from the field of ANNs. The idea is to input a signal into an RNN and use its complex dynamics to perform calculations. In order to understand the functioning of RC and at the same time the reason why it has become increasingly popular, it is impossible to avoid at least a basic comparison with classical neural networks, although the operation of the latter is not the subject of study of this work.

In a typical neural network, there are three main types of layers: input layer, hidden layer(s), and output layer. The input layer receives the initial data, which is then passed through the network to produce an output in the output layer. The hidden layer(s) exist between the input and output layers and play a crucial role in transforming and processing the information. The most common method to train a network like that, is a process called Back Propagation (BP). During that process the neural network learns by adjusting the weights and biases associated with the connections between nodes. By repeatedly adjusting the weights, the neural network can gradually improve its ability to make accurate predictions or classifications. The BP neural network has been widely applied in many fields, such as pattern recognition, function approximation and image processing etc.

However, the BP algorithm has some disadvantages, such as poor rate of convergence, sensitivity to initialization and getting stuck in local minimum easily. In the following

figure, one can see a basic representation of how such a network operates.

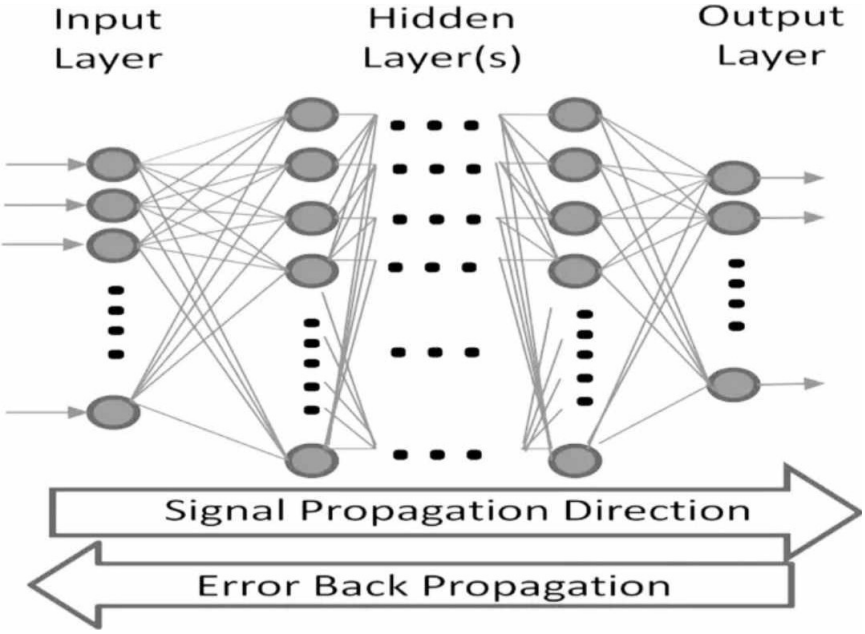


Figure 1.1: BP Network-Schematic Representation

It is easily perceptible that we have to deal with a very complex arrangement, which obviously leads the most times to a significantly time consuming learning process. As a result, there arises the need for a simpler method that can be implemented in much less time, while using remarkably fewer computational resources. It is worth mentioning at this point that it is not uncommon for training to take days, or even weeks, depending on factors like data size, network architecture, etc.

The reason behind that slow-paced learning process is mainly the quite large number of statistical weights between the connections of the nodes, which the algorithm needs to optimize. This could be the weakest part of BP networks that RC prevails over them. More specifically, RC does not require to adjust every individual weight, but on the contrary it is necessary to train the weights that represent connections which lead only to the output layer.

There is a multitude of advantages that make Reservoir Computing a promising approach for tackling complex temporal tasks efficiently and effectively. Some of these are the simple network structure, the fast training and of course the adaptability to diverse applications. The most common applications are natural language processing, classification, medical diagnosis, signal processing and time-series forecasting, with the last one being further discussed in the upcoming chapters of this thesis.

In the diagram below, one can observe the basic logic upon which such a network is constructed, as well as compare it to the network in the first diagram, with the goal of beginning to understand, even in a schematic manner, the advantages of the second network.

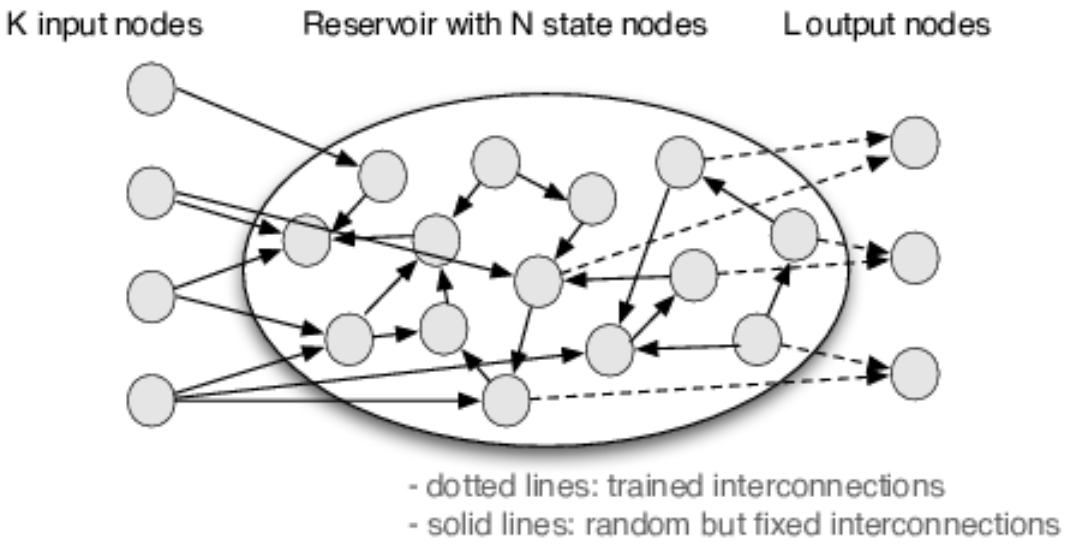


Figure 1.2: RC-Schematic Representation

1.1 Types of Reservoirs

Reservoir computing contains various types of computational frameworks that utilize recurrent neural networks known as reservoirs for processing temporal data. Echo State Networks (ESNs) are a widely used form of reservoir computing, where the reservoir network is randomly generated and sparsely connected. Liquid State Machines (LSMs) simulate the dynamics of a liquid, treating the reservoir as a continuous-time dynamical system. Time Delay Reservoir (TDRs) employ temporal delays into the dynamics of the reservoir, making it particularly suitable for capturing temporal patterns and dependencies in the input data. These diverse types of reservoir computing offer flexibility and power in handling temporal data, so they deserve a further analysis.

1.1.1 Echo State Networks (ESN)

Echo State Networks (ESNs) are composed by a recurrent layer called *reservoir* and an output layer called *readout*. The reservoir is randomly initialized and left untrained, while, the readout weights are trained through offline learning or online learning.

For each time step t, the model receives an input vector $u(t)$ through the input matrix W_{in}. The internal states vector $x(t)$ is generated by the reservoir's internal dynamics interacting with the input data. Subsequently, the matrix W_{out} calculates a readout from the state vector, generating the output vector $y(t)$. Optionally, this vector can be utilized as feedback to the reservoir for updating the internal states along with the vector $u(t+1)$ in the subsequent iteration. This process can be shown in the upcoming figure.

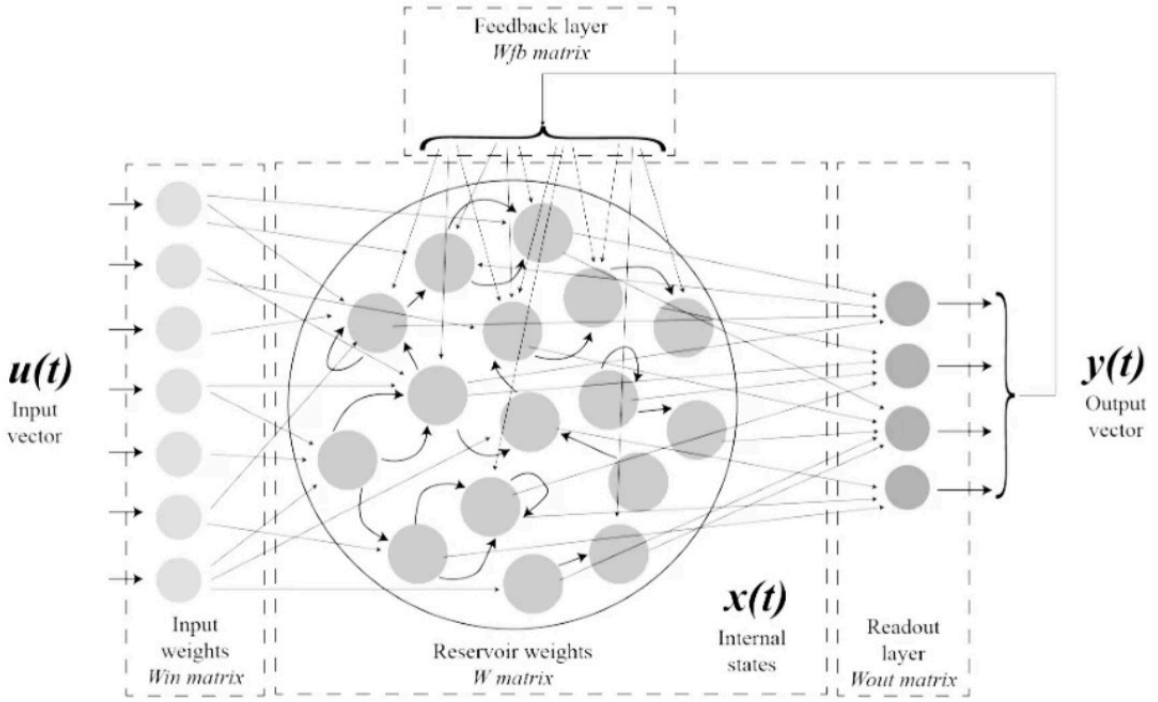


Figure 1.3: ESN-Schematic Representation

Since Echo State Networks are the type that will be of particular interest in this study, we will settle on this brief analysis for now and revisit *Chapter* **??** for further exploration.

1.1.2 Liquid State Machines (LSM)

Taking inspiration from the behavior of liquids in a container, Liquid State Machines (LSM) mimic the dynamic ripple effect generated by dropping a stone into tranquil water. In a similar manner, the input signals propagate through the network, generating cascades of activity within the nodes.

More explicitly, in an LSM, an extensive ensemble of interconnected nodes, also known as neurons, is harnessed for processing input signals. These nodes receive input from both external sources and other nodes within the network, giving rise to a complex and intricate web of connections. Through the interactions and information flow among the nodes, the input signals undergo intricate transformations, resulting in the emergence of spatio-temporal patterns of node activations.

The true power of LSMs lies in their ability to perform a diverse range of nonlinear computations on the input data. By utilizing linear discriminant units, the patterns of node activations can be effectively read and interpreted. This enables the LSM to execute a multitude of mathematical operations, custom-tailored to address specific tasks such as speech recognition or computer vision. A highly vivid depiction of the functioning of these networks is the image that follows.

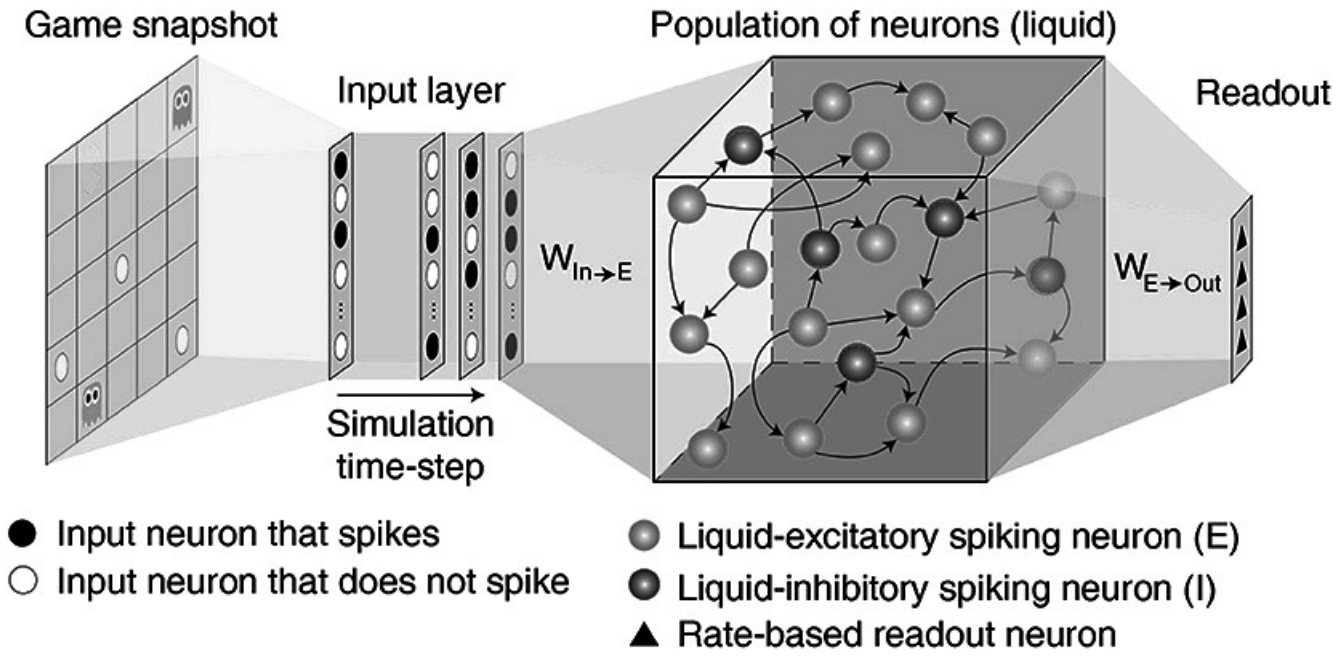


Figure 1.4: LSM-Schematic Representation[1]

LSMs have distinct disadvantages that differentiate them from other network architectures. Firstly, their relatively higher computational complexity and resource requirements. The simulation and training of large-scale LSMs can demand significant computational resources and memory capacity, making them less efficient in certain scenarios. Another notable disadvantage of LSMs is their complex and rich dynamics, while advantageous for capturing temporal dependencies, can make them more prone to instability and sensitivity to noise. This heightened sensitivity can potentially limit their robustness[2] and generalization abilities compared to other network types. It's important to consider these unique drawbacks when assessing the applicability of LSMs for specific tasks and environments.

1.1.3 Time Delay Reservoir (TDR)

A time delay reservoir (TDR), also called as delayed feedback reservoir (DFR), is a type of reservoir computing architecture that incorporates time-delay elements within the reservoir. In a TDR, the reservoir network is augmented with feedback loops or tapped delay lines that introduce time delays in the signal propagation. These time delays allow the reservoir to capture temporal dependencies and preserve the history of past inputs.

The purpose of integrating time delays in the reservoir is to enhance its ability to process time-dependent information and capture complex temporal patterns. By

[1]A visual depiction is provided, showcasing a captured moment from the Pacman game. This snapshot is subsequently transformed into five two-dimensional binary representations, each corresponding to the positions of Pacman, foods, cherries, ghosts, and scared ghosts. These binary representations are then flattened and combined together to form the representation of the game environment state.[PSR19]

[2]Regarding the robustness of a network, it is an aspect that has particularly concerned the author of this dissertation and will be extensively discussed later on.

introducing delays, the TDR can effectively handle sequential data and exploit the temporal dynamics present in the input signals.

The time delay elements in a TDR can be adjusted to control the memory length and the temporal resolution of the system. Longer delays can capture longer-term dependencies, while shorter delays allow for finer temporal resolution.

It is widely accepted that understanding the functioning of TDRs is not something that an unfamiliar reader could easily grasp from the outset. In my attempt to help the reader better understand the logic behind this arrangement, I suggest keeping in mind the following fundamental operational principles. Firstly, this configuration consists of a single physical node, with all the remaining (N-1 in total) nodes being virtual. Secondly, the virtual nodes are implemented through a line, known as the delay line. So, in the provided diagram one can perceive the straightforward yet sophisticated implementation of a TDR.

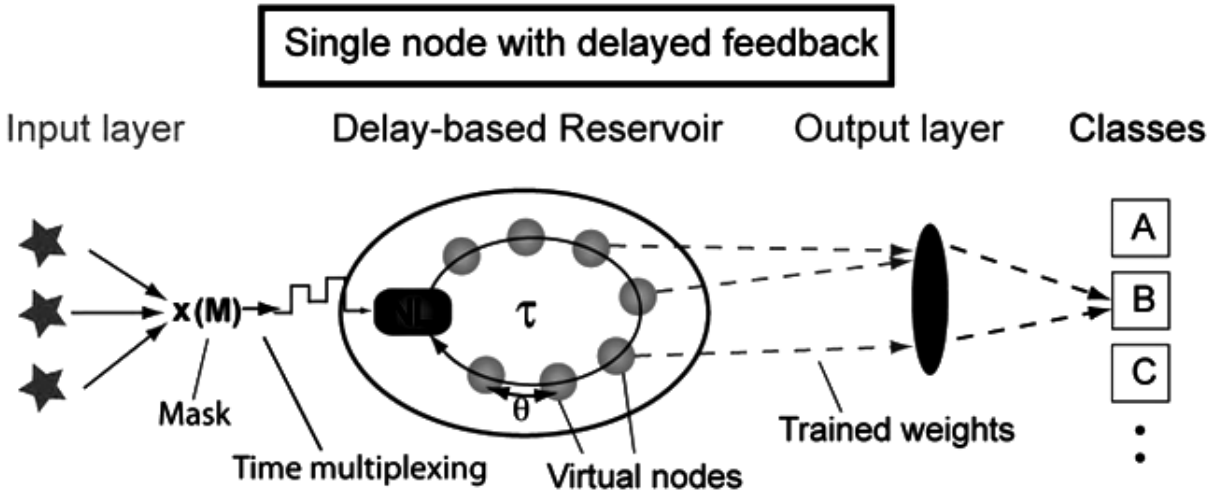


Figure 1.5: TDR-Schematic Representation [**soriano2015minimal**]

TDRs have been successfully applied to various tasks such as time series prediction, security, speech recognition, and control systems where the temporal nature of the data is essential. They provide a powerful framework for modeling and processing time-dependent information, leveraging the dynamics of time-delayed interactions within the reservoir network. Lastly, it is worth mentioning that the architecture of these networks appears to be ideal for implementation using simple electronic or optical systems.

1.2 Physical Reservoir Computing

Physical reservoir computing (PRC) is a computational framework that harnesses the dynamics of a physical system, typically a complex and nonlinear one, as a reservoir for performing computations. Instead of relying on sequential processing and explicit algorithms like traditional computing architectures, PRC takes advantage of the

intrinsic computational power of the physical dynamics to enable parallel and distributed computation.

Physical reservoir computing leverages the physical system itself as the reservoir, removing the need for a separate reservoir component. Various physical systems, including electronic circuits, optoelectronic devices, photonic systems, or biological systems, can be employed to implement the reservoir. These physical systems exhibit complex dynamics and possess inherent parallel processing capabilities, making them well-suited for reservoir computing applications.

Physical reservoir computing has demonstrated promise across multiple applications, such as time-series prediction, pattern recognition, signal processing, and control systems. It offers advantages in terms of simplicity, scalability, and noise robustness. Furthermore, the physical implementation of the reservoir presents potential benefits like low power consumption, fast computation, and compatibility with various physical platforms.

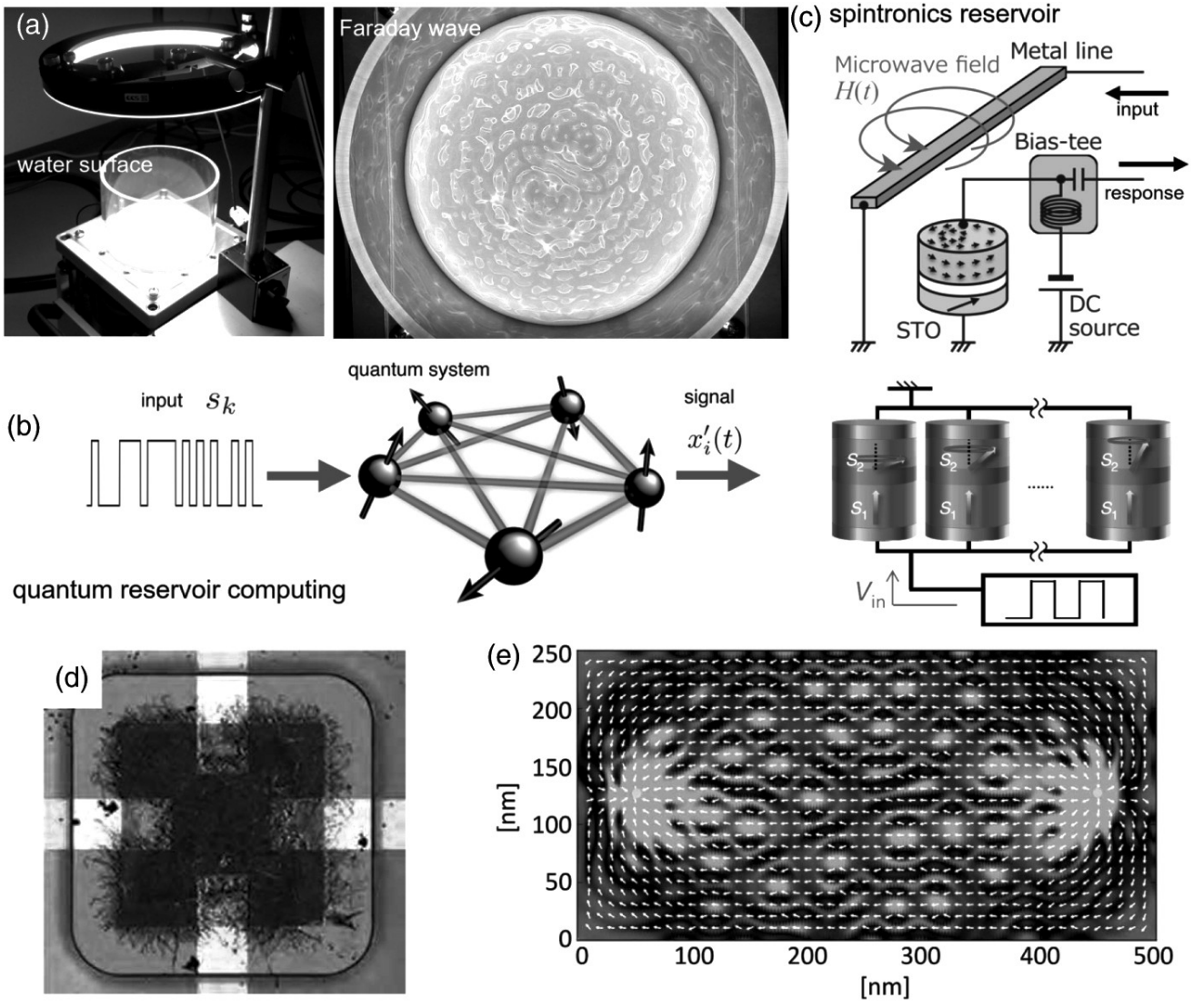


Figure 1.6: Variations of physical reservoirs.

(a) The physical liquid state machine proposed. It exploits the Faraday wave as a computational resource. (b) Quantum reservoir computing. It allows one to exploit disordered ensemble quantum dynamics as a computational resource. (c) Variations of the spintronics reservoir. The upper and lower diagrams show reservoirs, which exploit vortex-type spintronics and spatially multiplexed magnetic tunnel junctions respectively. (d) Complex Turing B-type atomic switch networks. The complex nanowire network extends throughout the device and is probed via macroscopic electrodes. (e) A skyrmion network embedded in frustrated magnetic films. The current path is visualized after the voltage is applied to the frustrated magnetic texture including Bloch skyrmions. [Nak20]

It is important to note that physical reservoir computing is an active area of research, with ongoing exploration of its full potential and practical limitations. Researchers continually strive to develop new physical systems, enhance training algorithms, and explore novel applications for this computational paradigm. Therefore, we will present three out of the many physical reservoir computing types.

1.2.1 Electronic RC

Electronic reservoir computing (ERC) is a specialized approach that focuses on utilizing electronic hardware to implement reservoir computing. The primary objective of ERC is to capitalize on the benefits offered by electronic systems, such as their built-in parallelism and the potential for faster processing speeds. By doing so, ERC aims to conduct efficient and high-performance computations.

In the context of ERC, the term ”reservoir” refers to a dynamic system that consists of interconnected electronic components, including capacitors, resistors, and operational amplifiers. These components combine to form a network with recurrent connections, leading to a sophisticated and intricate dynamic behavior. The primary function of the reservoir system is to serve as a computational platform that receives input data, processes it, and maps it to the desired output values.

1.2.2 Optical Reservoir

Optical reservoir computing (ORC) is an advanced approach to reservoir computing that capitalizes on optical systems for information processing. ORC takes advantage of the unique attributes of light, such as parallelism and high bandwidth, to execute computational tasks with efficiency.

Within ORC, the reservoir represents an optical system comprised of interconnected optical components like lasers, waveguides, and spatial light modulators. These optical elements form a network with recurrent connections, resulting in a reservoir that displays complex and diverse dynamical behavior.

Figure 1.7 illustrates the experimental arrangement for implementing optical reservoir computing. The setup comprises several key optical components, including phase-only Spatial Light Modulators (SLM), a scattering medium, and a camera.

The SLM plays a crucial role by encoding both the input vector $\mathbf{i}(t)$ with a dimension of D_{in} and the subsequent reservoir state $\mathbf{r}(t)$ with a dimension of D_{res} (total dimension

$D_{in} + D_{res}$). This encoding is achieved by manipulating the phase spatial profile of the light.

The scattering medium ensures that the encoded input vector and reservoir state undergo random linear mixing. This mixing can be seen as equivalent to performing linear multiplications with large, dense random matrices composed of independent and identically distributed (i.i.d.) random complex variables.

Lastly, the camera is responsible for performing a nonlinear readout of the complex field intensity, resulting in the next reservoir state $\mathbf{r}(t + \Delta t)$. This state is then sent back to the computer, which in turn displays it on the SLM along with a new input vector, initiating the iterative process. The upper and lower insets in Figure 1.7 provide examples of the images displayed on the SLM and captured by the camera, respectively.

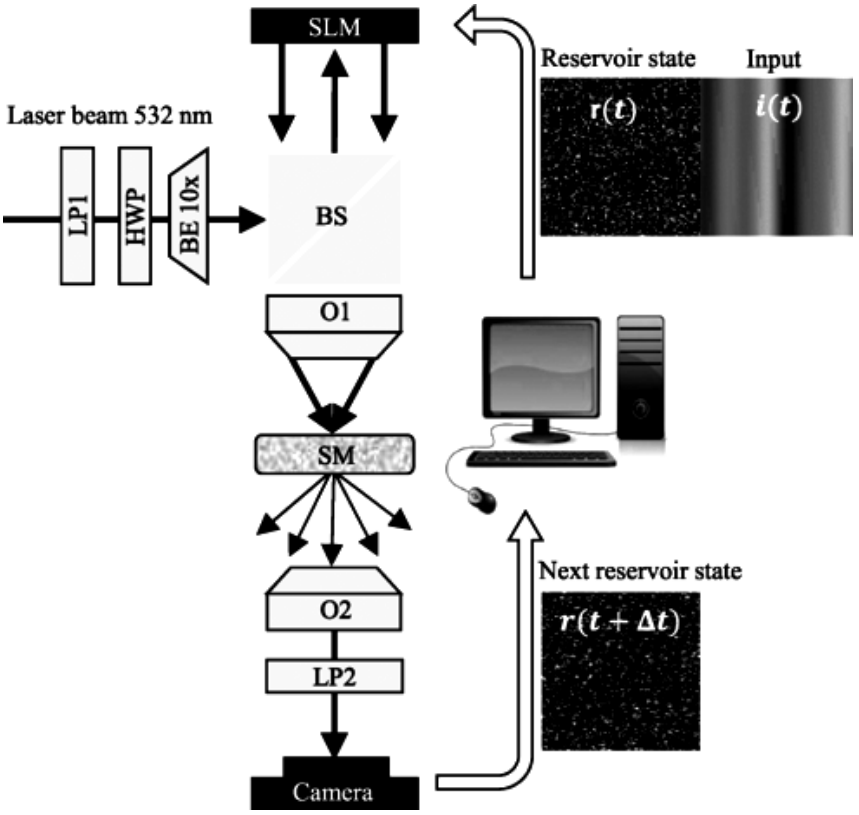


Figure 1.7: Experimental setup to perform an optical reservoir computing. The Spatial Light Modulator (SLM) receives from the computer the consequent input $\mathbf{i}(t)$ concatenated with the reservoir state $\mathbf{r}(t)$ and imprints it into the spatial phase profile of the reflected beam (see the upper inset as a typical example). The scattering medium (SM) provides a complex linear mixing of the whole encoded information. Finally, the camera performs a nonlinear readout for the next reservoir state $\mathbf{r}(t + \Delta t)$ (see the lower inset as a typical example), which is sent by the computer back to the SLM to be displayed with new input, and the process repeats. LP1-LP2: linear polarizers, HWP: half-wave plate, BE: beam expander, BS: beam splitter, O1-O2: objectives. [Raf+20]

1.2.3 Memristive Networks

In this chapter, we will explore memristive networks as the final subject of study. However, in order to proceed with explaining the operation of such a network, it would be wise to first refer to what a memristor is and what it actually does.

A memristor, conceptualized by Leon Chua in 1971 and experimentally verified in 2008, is an electronic device with two terminals that possesses a distinctive quality known as memristance. Coined by combining ”memory” and ”resistor”, the term ”memristor” signifies its capacity to retain and store data in the form of resistance. In contrast to standard resistors that uphold a consistent resistance, memristors have the ability to modify their resistance in response to the magnitude and direction of the electrical current flowing through them. There are two main categories into which memristors are classified based on their behavior.

Volatile memristors are characterized by their tendency to reset their resistance state when the power supply is interrupted or switched off. In simpler terms, the information stored in volatile memristors is temporary and necessitates refreshing or reprogramming whenever power is reestablished. These memristors are well-suited for applications that involve frequent modifications or updates to the stored information.

Non-volatile memristors exhibit the ability to preserve their resistance state even in the absence of a power supply. This means that the stored information remains intact and unaffected when power is restored. Similar to non-volatile memory devices like flash memory or hard drives, non-volatile memristors can retain data for extended periods without the need for a continuous power source. These types of memristors are particularly advantageous in applications that demand reliable and persistent storage of information.

Memristors, with their unique behavior, appear to be highly promising devices that could contribute to a multitude of applications. However, in reality, their response is not consistently stable from one instance to another. This behavior can be confirmed by referring to Figure 1.8.

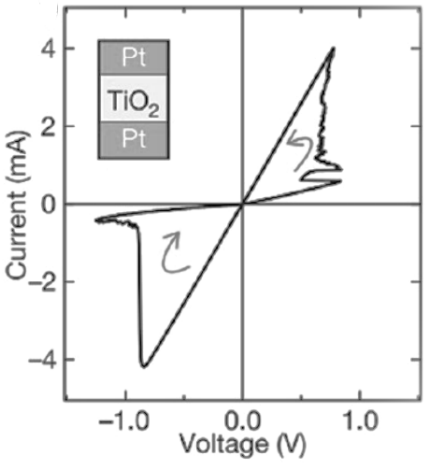


Figure 1.8: Characteristic current-voltage curve of a memristor. [Guo+20]

In order to interpret this behavior, it is paramount to mention that a memristor consists of a thin layer sandwiched between two electrodes, known as the memristive layer. Made of materials like metal oxide, this layer enables resistive switching. The bottom and top electrodes apply voltage or current to induce resistance changes. Therefore, imperfections within the memristive layer, such as vacancies or dopants, play a crucial role in enabling ion movement or charge redistribution, resulting in resistance modification.
Reminder: The slope of the graphical representation of the function $I = f(V)$ aligns with Ohm's law, where the conductance G is equal to $1/R$.

Even with these imperfections, memristors still possess a memory-like characteristic, enabling them to preserve information regarding the past electrical signals they have encountered. This attribute empowers them to store and retrieve various resistance states, thereby facilitating the creation of electronic synapses that emulate the behavior of biological neurons within neural networks.

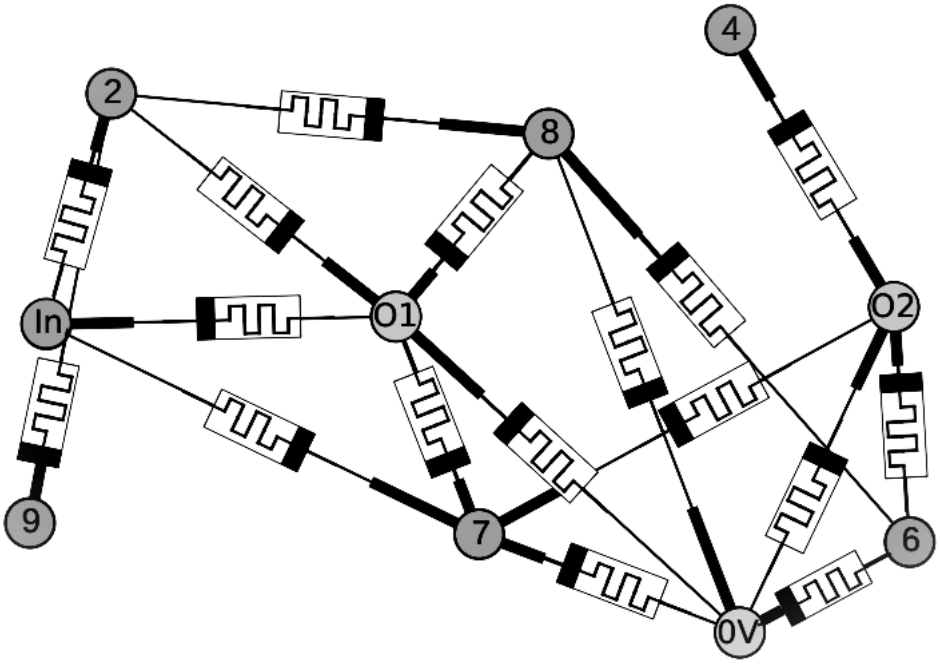


Figure 1.9: Example of a randomly assembled memristive network. The circles indicate nodes in which memristive devices (links between nodes) connect. In: input node, 0V: ground node, O1-O2: differential output nodes. The network size and device density is controlled by the number of nodes N and the in-degree K (number of connections per device). [Bür+15]

Our laboratory *(LaNSCom)* is deeply involved in the research of memristive networks, focusing on their study and optimization. For this thesis, we have exclusively utilized digital methods, employing computer assistance to implement the networks and conduct tests. Nevertheless, the analog implementation and performance continue to be a crucial area of investigation for our research, which could potentially serve as a continuation of this work in the future.

Chapter 2

A fully comprehensive analysis of ESNs

In the preceding chapter, we provided a brief overview of the three fundamental types of reservoir computing. The objective of this second chapter is to offer a more comprehensive and detailed explanation of the Echo State Networks. These networks are likely the most widely used among all types and serve as the primary focus of this research.

2.1 Operation

First of all, it is necessary to clarify that, unlike the other two types (LSMs,TDRs) where time is continuous, in ESNs, time is discrete. That is, the nodes of the reservoir are described by discrete states $x(n)$ and likewise, $u(n)$ represents the discrete states of the input signal, while $y(n)$ represents the output. Additionally, nodes are artificial neurons that implement a non-linear transfer function. It is true that the state update can be controlled by any sigmoidal function in theory. However, for consistency and in line with the prevailing literature in reservoir computing, we opt to use the $tanh$ function as the standard sigmoid function.

Taking into consideration the points we discussed earlier, below is the most generall equation governing the functioning of the reservoir, commonly referred to as the *Recursive Reservoir Equation*.

$$\boxed{x(n+1) = (1-a)x(n) + a\tanh(Wx(n) + W^{in}u(n+1) + W^{fb}y(n) + \upsilon(n))} \tag{2.1}$$

The current state of each neuron or node in the reservoir is determined by its previous states, in accordance with equation 2.1. In this equation, the parameter α appears, referred to as the leaking rate, and its impact will be explained in the section 2.2, which is dedicated to hyperparameters.

We also encounter the matrices W^{in} and W, where W^{in} is the matrix of input weights, and W is the matrix of reservoir weights. They have dimensions [3] $(N \times K)$ and

$(N \times N)$, respectively. It is worth noting that both of these matrices contain random yet fixed and non-trainable weights. We can also define the dimensions of the other vectors, where $u(n) : (K \times 1)$, $x(n) : (N \times 1)$, and $y(n) : (L \times 1)$. Thus, it is evident that the u, x, and y at each time step n take the form of a column vector, with $n = 0, 1, 2, ..., M$, where M represents the total number of time steps in the dataset.

There is also a part of equation 2.1 that we have not explained, and that is the term $W^{fb}y(n) + \upsilon(n)$. This term is an optional component, where W^{fb} represents the randomly generated feedback weight matrix, and $\upsilon(n)$ is the noise function. Most of the time, this term is left out since its positive impact on prediction accuracy is not always guaranteed and depends on various factors, such as the nature of the problem, for example.

As reiterated, the W^{out} matrix, which is the matrix that represents the output weights, holds the key to the operation of ESNs. It is the only matrix that undergoes training and is not fixed, setting it apart from the rest of the components in the network. Consequently, the following reasoning guides us towards the desired outcome, which is the determination of the W^{out} matrix.

Let's assume that the Mean Squared Error (MSE) is given from the above equation:

$$MSE = \frac{1}{M}\sum_{i=1}^{M}(y^{out}(n) - y^{target}(n))^2 \tag{2.2}$$

We also know that reservoir's output is a linear combination of the total activations, thus:

$$Y = W^{out}X \tag{2.3}$$

supposing that Y and X represent matrix forms of the reservoir output $y(n)$ and reservoir states $x(n)$, respectively.

Combining equations 2.2-2.3, we get a different expression for MSE, which is:

$$MSE = (W^{out}X - Y^{target})^2 \tag{2.4}$$

The ultimate goal is to minimize the absolute MSE as much as possible, ideally aiming to reduce it to zero. In such a case,

$$W^{out}X = Y^{target} \tag{2.5}$$

There are conventional and widely accepted methods to solve equation 2.5. Firstly, someone could argue that

$$W^{out} = Y^{target}X^{-1} \tag{2.6}$$

[3] n: time, M: total time steps in data set, N: number of total nodes, K: length of input time series at specific time n, L: length of output time series at specific time n

But, we have to take into consideration that $u(n)$ is usually larger than the size of the reservoir, something that makes overdetermined our system of linear equations, described in expression 2.3.
Unfortunately, taking the inverse of an overdetermined matrix could possibly lead to unstable solutions.

It seems that the most universal and stable solution to equation 2.5 is likely ridge regression, which is also known as regression with Tikhonov regularization:

$$W^{out} = Y^{target} X^T (XX^T - \beta I)^{-1} \tag{2.7}$$

where β is a regularization coefficient and I is the identity matrix. Certainly, this is by no means a trivial method, but it can definitely relieve us from inexpediencies, such as the one mentioned earlier.

Nevertheless, if someone still seeks a straightforward solution to 2.5 but without the issues that arose from the previous approach, then they can apply the *Moore-Penrose pseudoinverse method*. More specifically, the final solution is

$$\boxed{W^{out} = Y^{target} X^{\dagger}} \tag{2.8}$$

where $X^{\dagger}$ is the Moore-Penrose pseudoinverse of X. One the one hand, this calculation typically exhibits high numerical stability. On the other hand, dealing with large design matrices X can be memory-intensive and costly.

2.2 Hyperparameters

After providing an explanation of how we reach the determination of the matrix containing the output weights (even without delving into complex mathematical concepts), we can now establish some vital parameters that significantly influence the reservoir's functioning.

- **Leaking Rate**

  The α parameter we saw in eq. 2.1, also known as the *leaking rate* is liimited to a value between 0 and 1 and mainly dictates how quickly the input influences or seeps into the reservoir. The leaking rate is generally configured to mirror the pace of the dynamics that the reservoir is trying to replicate from y^{target}. A leaking rate of 0 means that the reservoir's state completely ignores the new input, effectively only relying on its internal dynamics. On the other hand, a leaking rate of 1 means that the new input fully replaces the previous state, resulting in the reservoir discarding any prior information. This phenomenon becomes readily comprehensible from equation 2.1. When α is set to 1, the term $x(n)$ is entirely eliminated from the equation, while at the same time, the sigmoid function's impact is maximized. Conversely, for $\alpha = 0$, the opposite occurs. The reservoir's memory and information processing capabilities can be notably affected by the

leaking rate. Skillful adjustment of this parameter in Reservoir Computing models can lead to enhanced performance across different tasks.

- **Spectral Radius**

  The spectral radius ρ holds paramount importance in the echo state network, as it governs the reservoir's *echo state property*, a concept we will further discuss in section 2.3. It is characterized as the maximum absolute eigenvalue of the reservoir connection matrix W and directly influences the reservoir's memory length. A larger spectral radius results in an extended memory of the input history. Typically, the spectral radius should be adjusted to optimize reservoir performance, starting with a benchmark value of $\rho = 1$ and exploring other ρ values close to 1. When modeling systems that rely on recent input history, a smaller ρ is recommended, while a larger ρ is preferable when a more extensive memory is needed.

- **Regularization Parameter**

  In the context of ESNs and Ridge Regression, the lambda (λ) parameter serves as a tuning parameter for regularization. It plays a crucial role in striking a balance between closely fitting the training data and managing the model's complexity. When λ is set to zero, there is no regularization, potentially leading to overfitting the data. Conversely, increasing λ introduces a penalty term, encouraging smaller values for the model's parameters. This regularization mechanism helps prevent overfitting and enhances the network's ability to generalize well to unseen data. To conclude, λ is a pivotal hyperparameter in ESNs when applying Ridge Regression to the output weights, contributing to improved model robustness and predictive performance.

- **Reservoir Size**

  The reservoir's size plays a vital role in Reservoir Computing, significantly affecting its memory and information processing capabilities. A larger reservoir allows for the capture of intricate patterns and dynamics in data, but it comes with increased computational costs and longer training times. On the other hand, a smaller reservoir is computationally more efficient but may have limitations in memory and processing capacity. Careful consideration of the reservoir size is essential to strike the right balance between performance and computational efficiency in Reservoir Computing applications.

Above, we briefly mentioned the main parameters in reservoir computing and their impact on the accuracy of our computations. We also referred to the range within we expect their values and the rationale behind varying them. However, these are based on what is commonly encountered in the literature as guidelines and are not a one-size-fits-all solution for all cases. For example, it is not guaranteed that increasing the size of the reservoir will always result in improved performance. We are currently ceasing the analysis regarding how parameters influence the reservoir's performance.

We will resume this discussion in Chapter 4, where we will share our own research findings.

2.3 The Echo State Property

The echo state property in ESNs requires the hidden layer's internal states (reservoir) to remain stable during the information propagation process. In more straightforward terms, the network should not exhibit chaotic behavior or have internal states that vanish over time. Instead, it should sustain a consistent and reliable set of internal activations as it processes data. To achieve the echo state property, several design principles are followed when building an ESN, like reservoir initialization, scaling of connections etc. In practical applications though, meeting the condition $\rho < 1$ ensures the fulfillment of the echo state property. In certain cases, the echo state property can still be maintained even when $\rho \geq 1$, particularly when there is a non-zero input. However, using significantly large ρ values may drive the reservoir into chaotic and unpredictable behavior, leading to a violation of the echo state property.

Chapter 3

Description of the Problem and the new Software

After having thoroughly introduced the theoretical aspects of reservoir computing and its broad applications, as well as delving into the specific functioning and fundamentals of ESNs, we can now progress to a more practical and technical chapter in this work. Consequently, there will be a complete explanation of both the operation and potential of the program we employed. The aim of this chapter is to ensure that the reader fully comprehends how the program functions and can independently set up an executable script to run it. Furthermore, we will provide a brief explanation of the problem we faced through this work.

3.1 Vosvdra System

The Vosvdra System is an autonomous system of three differential equations that describes a macroeconomic model. This specific system exhibits chaotic behavior for certain values of the control parameters. Below, we can explore the equations that constitute the system.

$$\frac{dS}{dt} = aY + pS(k - Y^2)$$

$$\frac{dY}{dt} = u(S + F) \tag{3.1}$$

$$\frac{dF}{dt} = mS - rY$$

where S(t) stands for the savings of households, Y(t) is the Gross Domestic Product, F(t) is the foreign capital inflow and a,p,u,m,r are positive control parameters.

Below you can observe the shape of the signal of the chaotic time series that we obtain from the Vosvdra system, on a scale of 1 to 5.

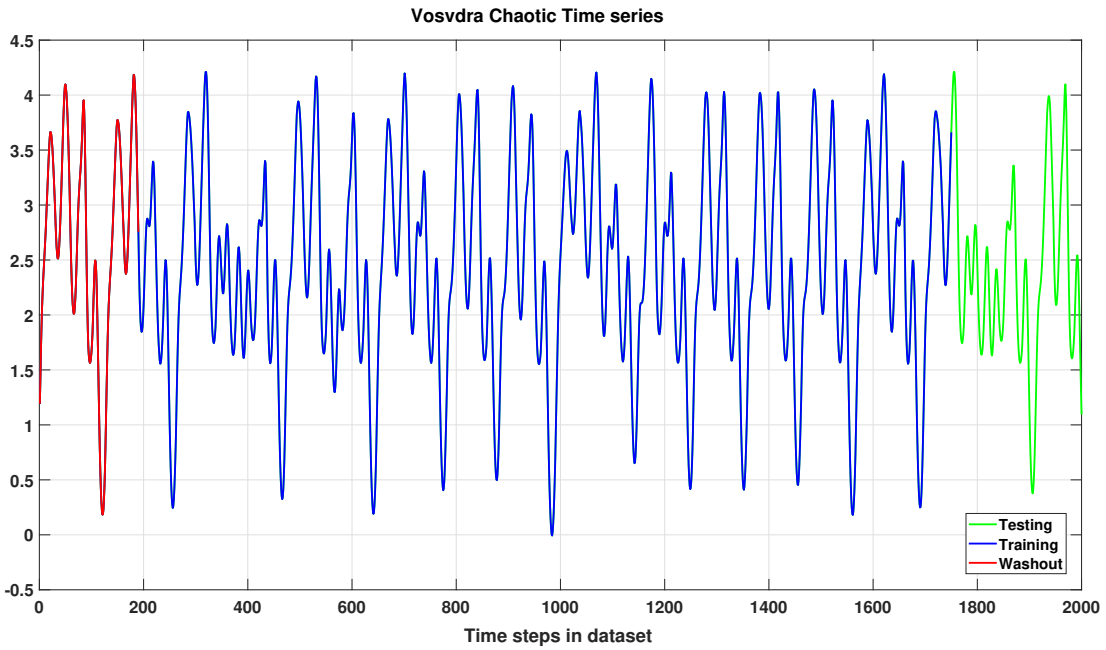


Figure 3.1: The y-axis represents the values of a variable

Furthermore, below we can see the graphical representation of the autocorrelation of the time series.

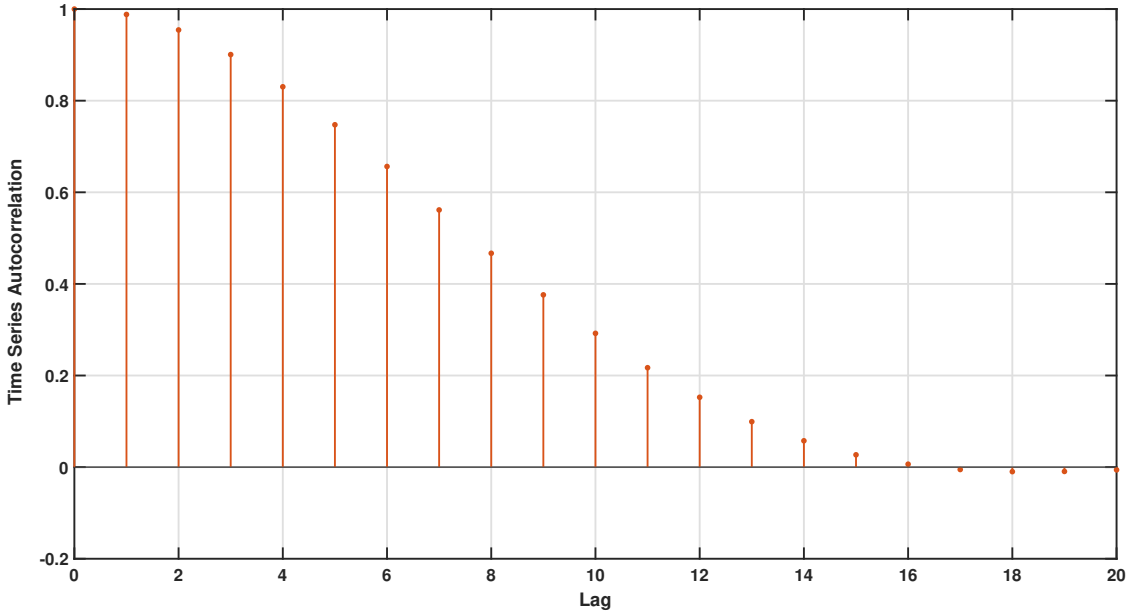


Figure 3.2: The x-axis represents the time steps

The self-correlation represents the degree of similarity between a given time series and a delayed version of itself over successive time intervals. Self-correlation measures the relationship between the current value of a variable and its previous values.

As easily perceivable, the autocorrelation becomes zero within the first 16 time steps. In this particular study, we will attempt to make predictions for 39 future time steps, which means we have to address a challenging problem, as we aim to forecast over twice the time window of autocorrelation nullification.

3.2 The MultiESN software

In this specific subsection, we will describe the MultiESN *[Ant23]* software, a tool that has been developed by Dr. *Ioannis Antoniades*. It is important to mention that this tool remains unpublished until the completion of this study. As depicted in the following figure, this software can implement these three topologies:

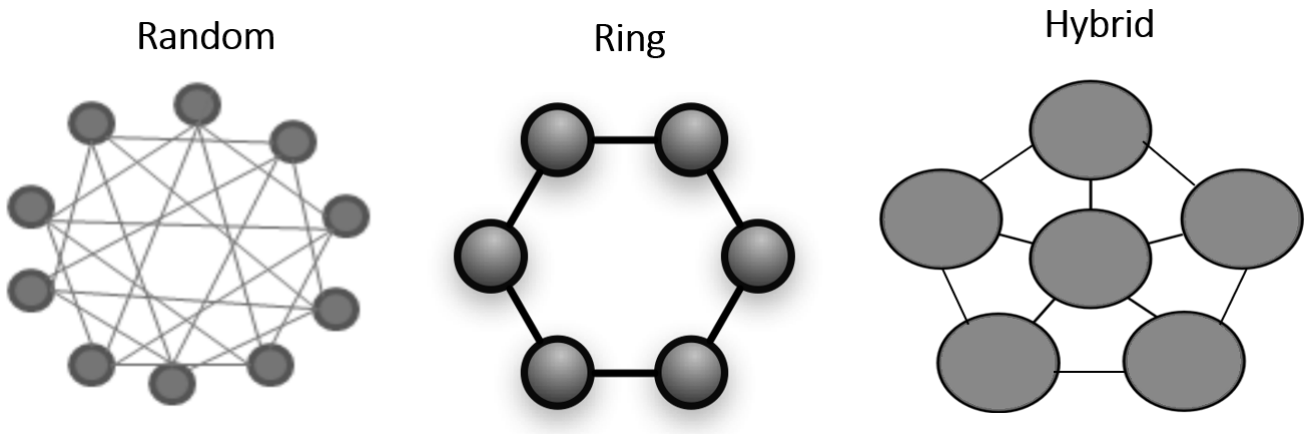


Figure 3.3: Description of the new ESN model: MultiESN

The software provides the user with additional options regarding the well-known ways of connecting the nodes in the reservoir, as well as the way of structuring and adding the input signal and output signal:

1. The model implements multilayered topologies, not necessarily deep.
2. Multi-layer architecture, with the ability to change the connectivity within and between layers.
3. Flexibility in determining the connection of both input and output with the reservoir.

So, having discussed the topologies implemented by this specific program, we can now focus on how to set up a time series prediction problem using MultiESN. In more detail,

we employ the *"Takens-embedding"* [Noa91] philosophy, which essentially creates delay matrices from both the input signal **u** and the target signal **y**. Here is how this works:

- Each column of the input matrix corresponds to the vector introduced into the reservoir at a given time t.
- The first three rows of column t hold the current value u_t of the input signal vector, the next triplet holds the value from the previous time step u_{t-1}, and the subsequent triplet holds the value from two time steps back u_{t-2}, and so on.
- Each column of the target matrix corresponds to the vector extracted by the reservoir at the same time step t. The first row is for the current time step y_t, the second row for the next one y_{t+1}, the third for the subsequent y_{t+2}, and so on.

To make this process more easily understandable to the reader, we include screenshots directly from the MATLAB file in which both the input signal and the output signal are visible.

	1	2	3	4	5	6	7	8	9
1	1.1923	1.4288	1.6694	1.8750	2.0422	2.1742	2.2885	2.3833	2.4707
2	2.0720	2.2851	2.4872	2.6473	2.7625	2.8310	2.8522	2.8185	2.7347
3	1.9369	1.2896	0.5544	-0.1829	-0.9135	-1.6263	-2.3716	-3.0535	-3.6430
4	0.9488	1.1923	1.4288	1.6694	1.8750	2.0422	2.1742	2.2885	2.3833
5	1.8299	2.0720	2.2851	2.4872	2.6473	2.7625	2.8310	2.8522	2.8185
6	2.5644	1.9369	1.2896	0.5544	-0.1829	-0.9135	-1.6263	-2.3716	-3.0535
7	0.7134	0.9488	1.1923	1.4288	1.6694	1.8750	2.0422	2.1742	2.2885
8	1.5605	1.8299	2.0720	2.2851	2.4872	2.6473	2.7625	2.8310	2.8522
9	3.1557	2.5644	1.9369	1.2896	0.5544	-0.1829	-0.9135	-1.6263	-2.3716

(α′) Input matrix

	1	2	3	4	5	6	7	8	9
1	1.1923	1.4288	1.6694	1.8750	2.0422	2.1742	2.2885	2.3833	2.4707
2	1.4288	1.6694	1.8750	2.0422	2.1742	2.2885	2.3833	2.4707	2.5612
3	1.6694	1.8750	2.0422	2.1742	2.2885	2.3833	2.4707	2.5612	2.6510
4	1.8750	2.0422	2.1742	2.2885	2.3833	2.4707	2.5612	2.6510	2.7508
5	2.0422	2.1742	2.2885	2.3833	2.4707	2.5612	2.6510	2.7508	2.8613
6	2.1742	2.2885	2.3833	2.4707	2.5612	2.6510	2.7508	2.8613	2.9801
7	2.2885	2.3833	2.4707	2.5612	2.6510	2.7508	2.8613	2.9801	3.0959
8	2.3833	2.4707	2.5612	2.6510	2.7508	2.8613	2.9801	3.0959	3.2141
9	2.4707	2.5612	2.6510	2.7508	2.8613	2.9801	3.0959	3.2141	3.3311

(β′) Output matrix

Figure 3.4: Direct screenshots of the MATLAB file

The last thing that is deemed necessary before proceeding to the most important chapter of the work, which is the results section, is to provide brief definitions of certain control parameters and new metrics that will be used along the way.

- **Delay:** It is the number of time steps in the past at a given time t that we load into the input of the reservoir.
- θ **(Timeweighting):** The θ weighted average of any set of values $x_0, x_1, ..., x_n$ is the weighted mean defined as follows:

$$< x >_\theta = \frac{\sum_{i=0}^{n} x_i e^{-i/\theta}}{\sum_{i=0}^{n} e^{-i/\theta}} \tag{3.2}$$

Where θ is a parameter that determines how rapidly the exponential weighting factor of the value xi decreases as the index i increases. The smaller the θ, the less future values are weighted compared to values closer to the present (i.e., for i=0). As θ approaches ∞, all x_i values are weighted uniformly, and the equation approaches the simple mean average.

- **Metametric**: Let M be a metric evaluating the performance of the reservoir computing system, for which we have values M_i at each time step i in the time series. Then, we define the meta-metric $\hat{M}(a_+, a_-, \epsilon)$ of M as the quantity:

$$\hat{M}(a_+, a_-, \epsilon) = \frac{\sum_{M_i \geq \epsilon} M_i e^{a_+(M_i - \epsilon)} + \sum_{M_i < \epsilon} M_i e^{a_-(M_i - \epsilon)}}{\sum_{M_i \geq \epsilon} e^{a_+(M_i - \epsilon)} + \sum_{M_i < \epsilon} e^{a_-(M_i - \epsilon)}} \tag{3.3}$$

Where ϵ is a parameter that determines which value of M is weighted with a unit coefficient, while a_+ and a_- are exponential parameters that determine the weighting of values of M that are greater or less than ϵ, respectively. Essentially, the metametric represents a "weighted" average of different values of the metric at each time step, where the weighting factors increase exponentially with the distance of the value of M from the "baseline" value ϵ.

Chapter 4

Results

In this final and most important chapter, we will report all the results of the experiments we conducted throughout this study. However, before doing so, it is extremely important to mention that we could not have achieved the following results without the invaluable assistance of the *Aristotle* high-performance computing cluster of the Aristotle University of Thessaloniki. This cluster enabled us to simultaneously run various independent tasks, significantly accelerating the entire process.

4.1 Hyperparameter Optimization

First we begin with the optimization of the λ ridge regression parameter.

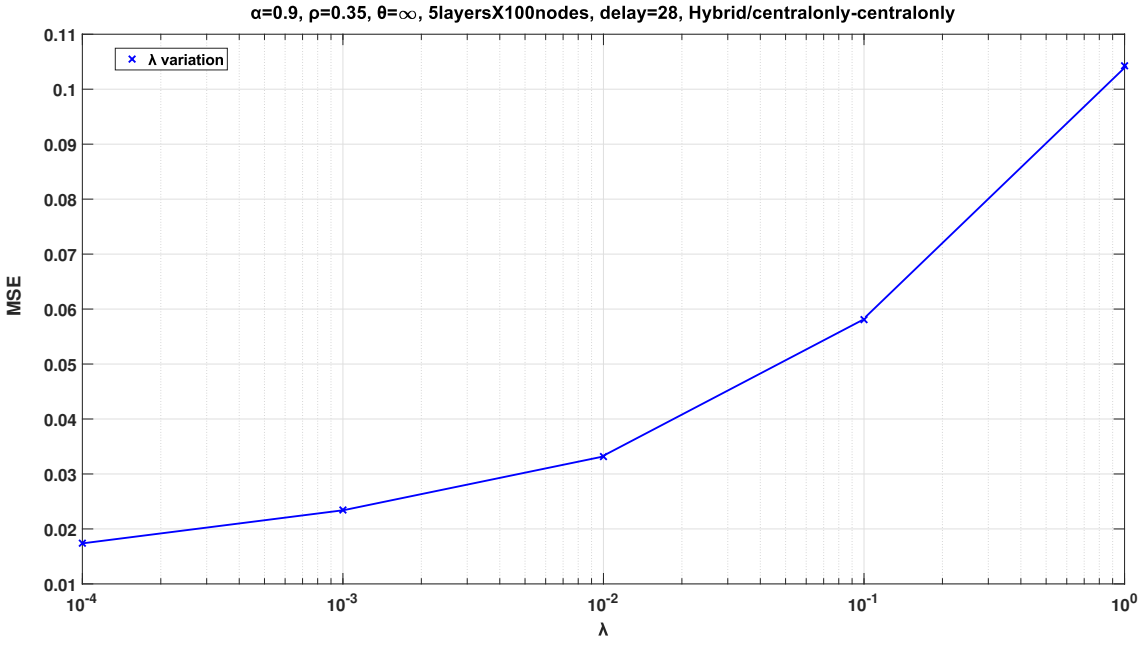


Figure 4.1: The x-axis is in logarithmic scale

In creating the particular chart, we employed λ values ranging from 0.0001 to 1. It is apparent that as λ rises, there is a notable increase in MSE, especially when approaching unity. In generall, bibliography mentions that the smaller the value of λ is, the more likely it is to lead to overfitting. However we did not face that issue in our case and as a result from now on we will either set the ridge regression parameter to zero, or we will use very small values.

We proceed with optimizing the hyperparameter α-leaking rate. We present four experiments precisely conducted under identical conditions but with different topologies.

In the first diagram, we have layers adhering to the hybrid topology. In this specific case, information between the layers is transferred only through the central nodes.

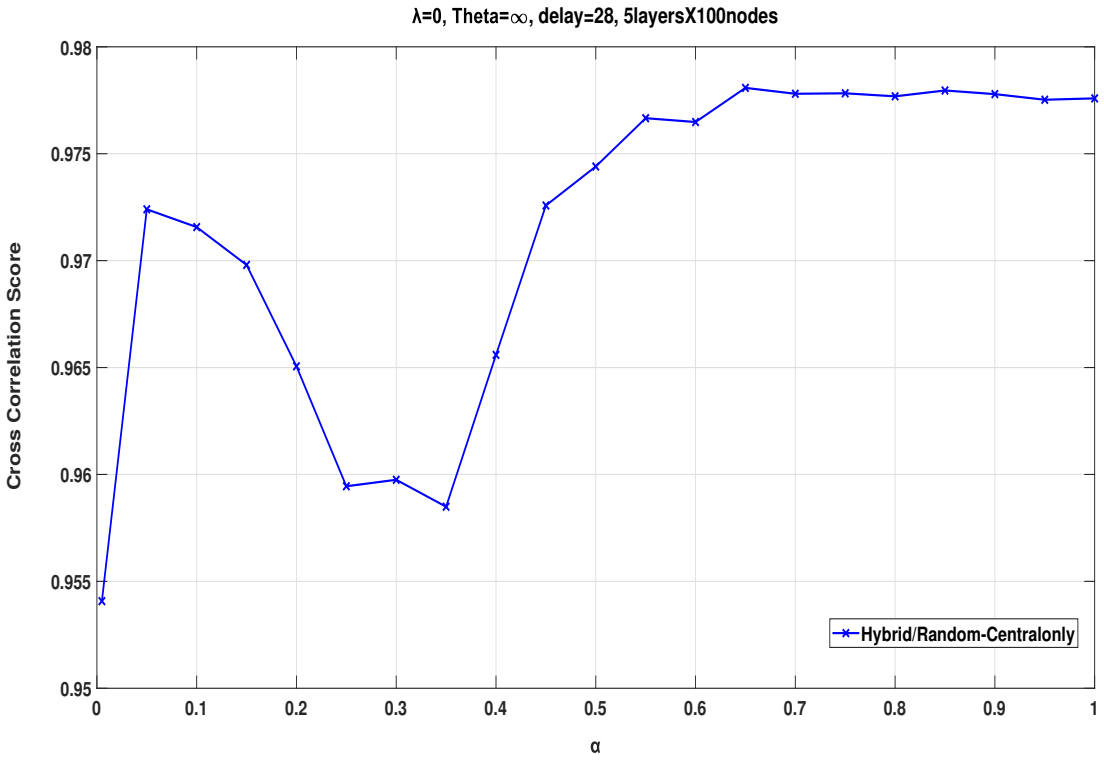


Figure 4.2: α variation/Hybrid-Centralonly

In the second diagram, the nodes of the layers are arranged in a ring structure, while the communication between the layers is random.

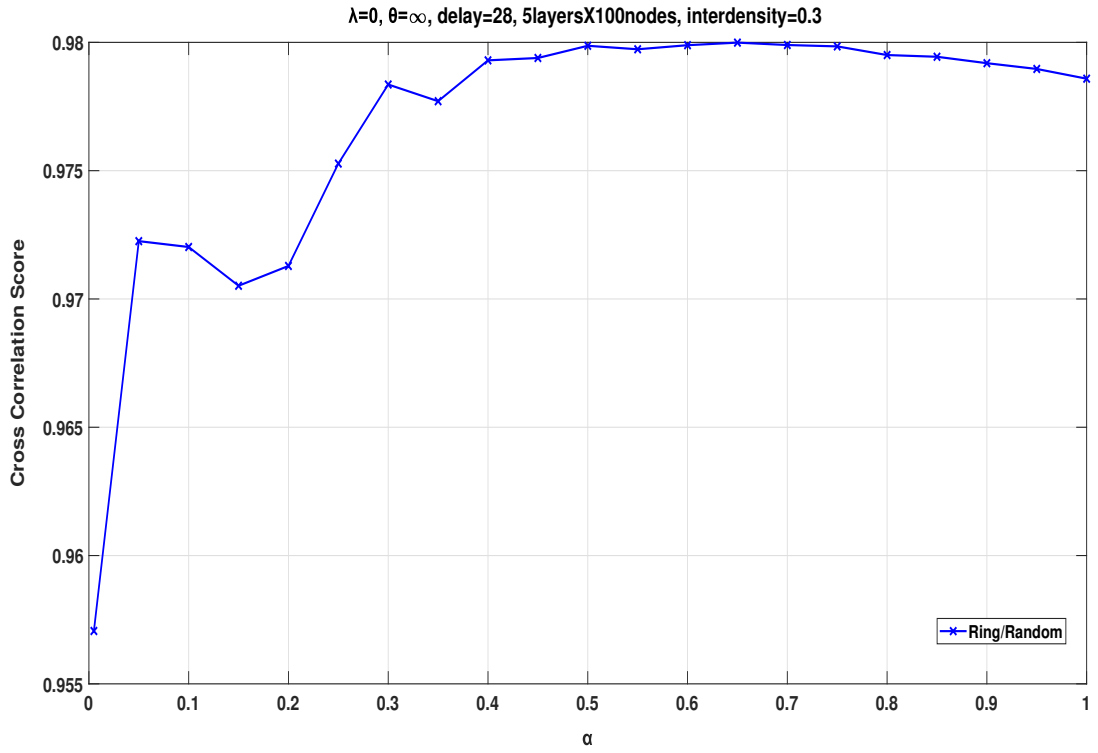


Figure 4.3: α variation/Ring-Random

In the third diagram, we have a hybrid internal structure with random interconnections.

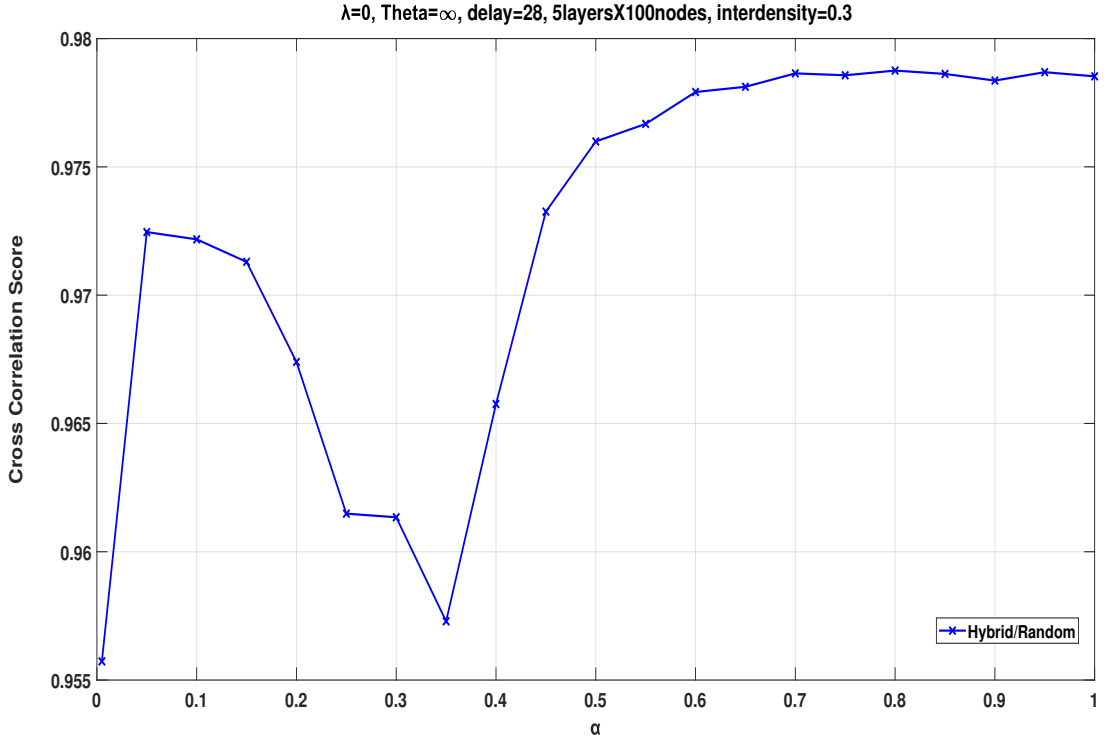


Figure 4.4: α variation/Hybrid-Random

In the fourth and final diagram, both internally and externally, nodes within the layers communicate randomly with each other, with probabilities determined by corresponding densities, namely the parameters intra and inter density.

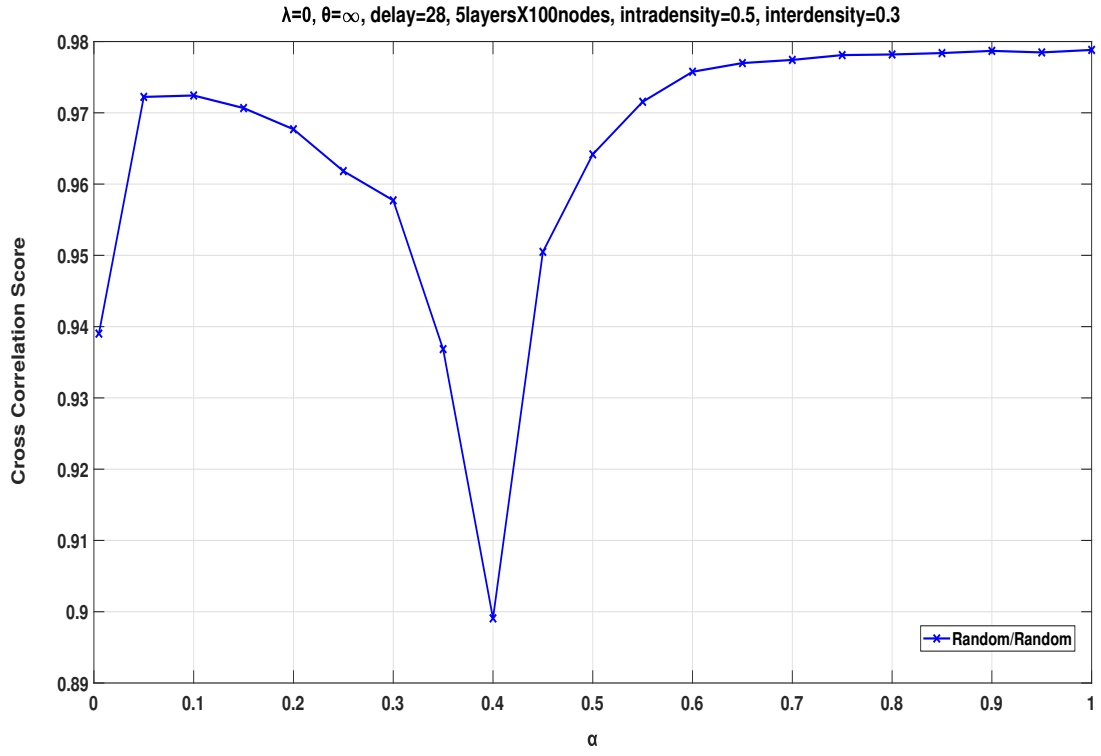


Figure 4.5: α variation/Random-Random

Observing these four diagrams, one could argue that there is not necessarily an ideal value for α, as a wide range of α values yield satisfactory results. However, it's easier to identify a "problematic" range for α, which extends to slightly lower values, primarily between 0.2-0.4. Generally, for this specific problem, it is safe to use values greater than 0.6 for that parameter.

It is worth noting that the specific diagrams are not presented randomly but in a hierarchical order. We begin with the implementation at the highest order, Hybrid/Centralonly and conclude with the one exhibiting the most disorder, Random/Random, where all connections are entirely random. This analysis is valuable in cases where one is interested not only in the digital but also in the analog implementation of networks.

We continue by repeating the exact same process to optimize the parameter ϱ. To achieve this, we can now choose a value for α that belongs to the safe range and keep it constant.

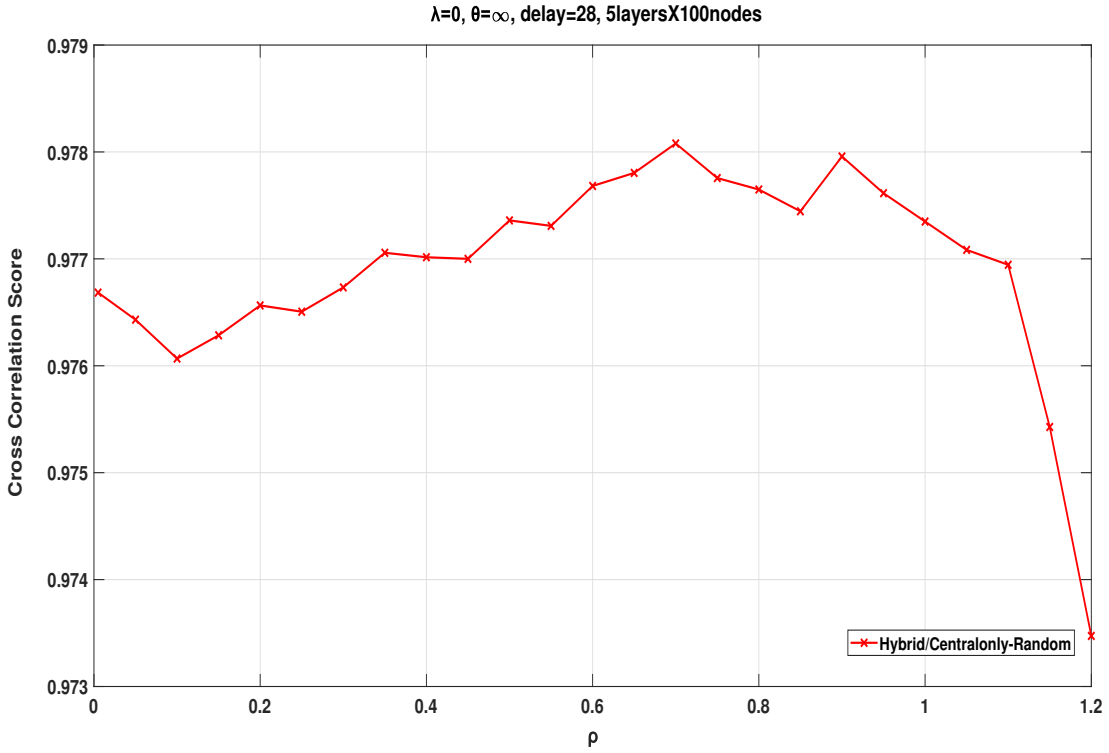


Figure 4.6: ρ variation/Hybrid-Centralonly

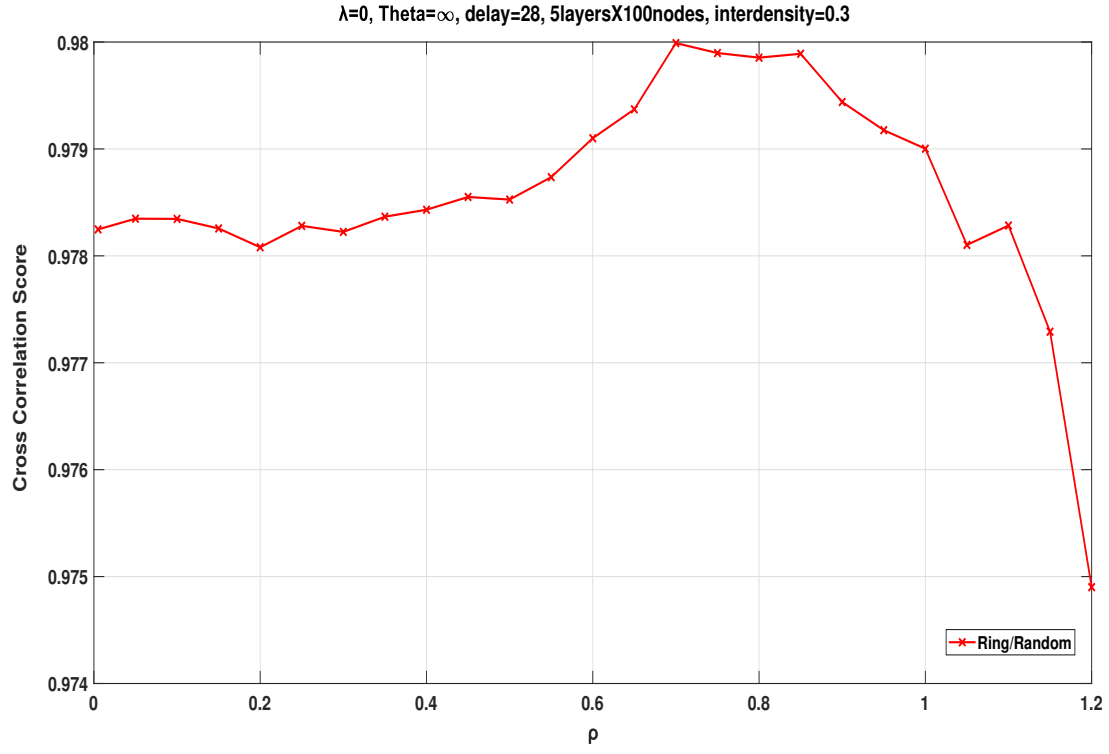


Figure 4.7: ρ variation/Ring-Random

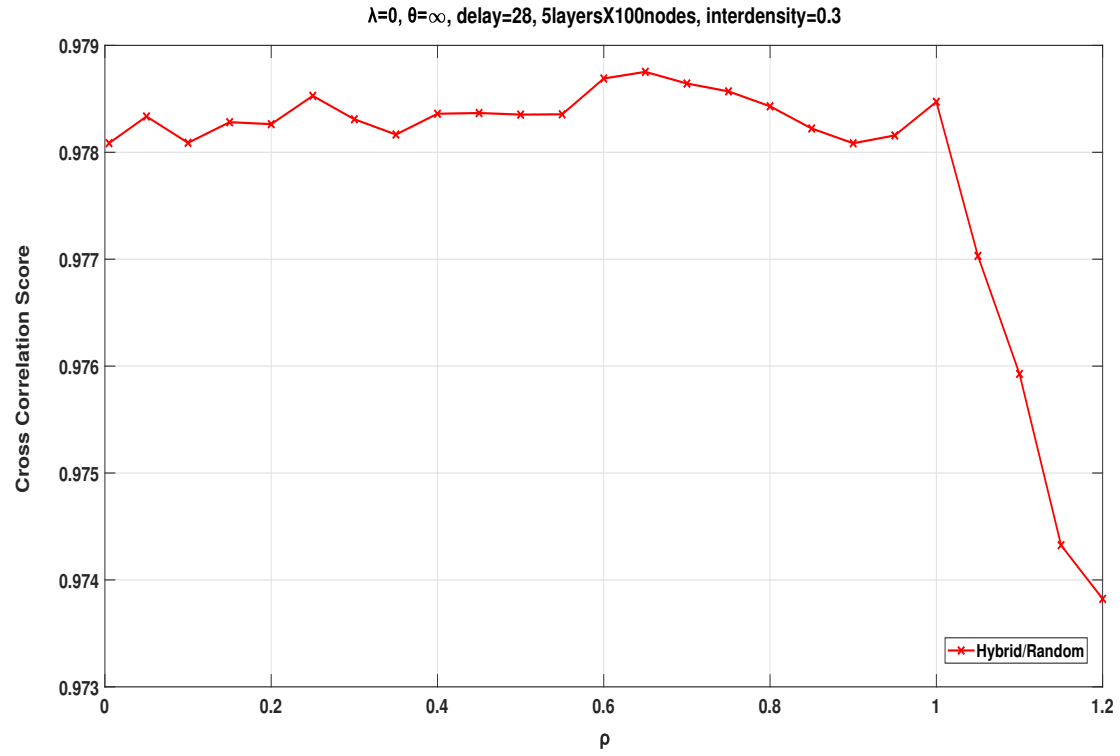


Figure 4.8: ϱ variation/Hybrid-Random

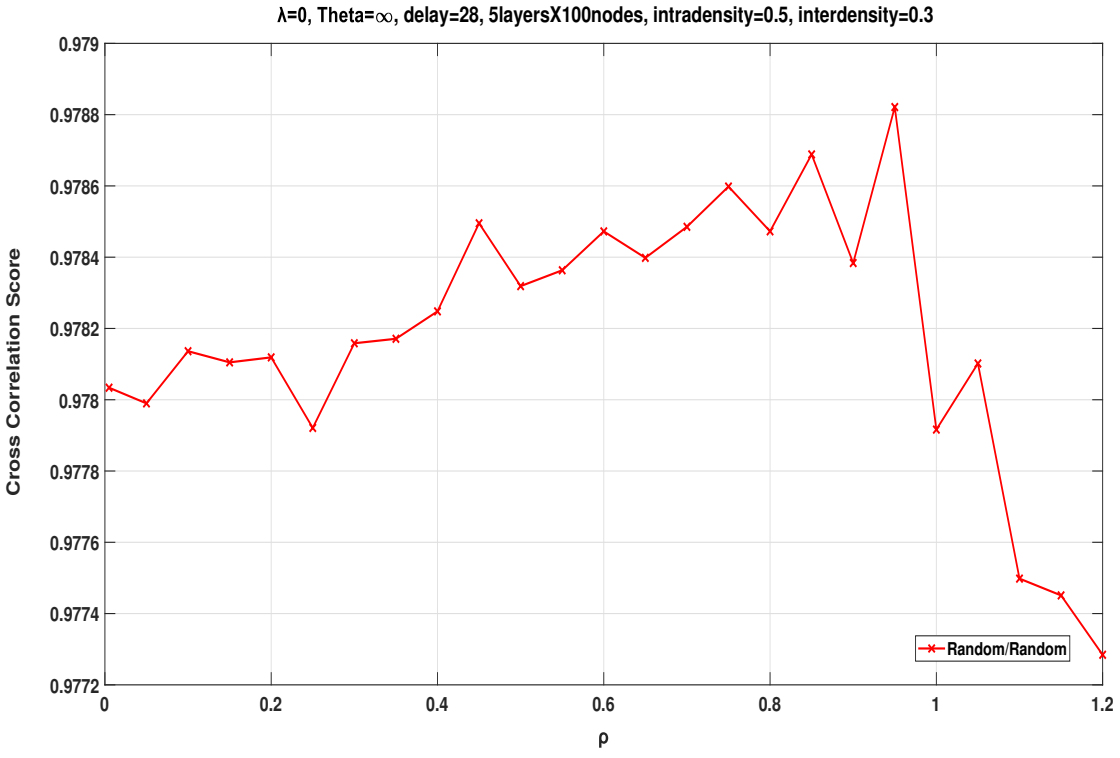


Figure 4.9: ϱ variation/Random-Random

Finally, regarding the parameter ϱ, which represents the spectral radius of the reservoir, we can safely conclude that it is not a highly influential parameter for the average performance of the reservoir in this particular problem. This is due to the fact that the variations are very small. What is certain, however, is that the user has no apparent reason to experiment with values greater than one, even though the literature allows for it, as these values lead to lower scores.

Note: Many of the diagrams provided above have a different metric on the y-axis than the ones defined so far, which is the Cross Correlation Score. Therefore, the necessary information is given below to specify this particular metric.

$$X_{cor} = \frac{\sum_{i=1}^{N} (y_i, y_i^{target})}{N} \tag{4.1}$$

where y_i is the output vector of the system at time i, y_i^{target} is the corresponding target value, and N is the length of the time series.
ϱ(A, B) is the correlation coefficient between lists A and B of length m:

$$\rho(A, B) = \frac{1}{m-1} \sum_{j=1}^{m} \frac{(A_i - \mu_A)(B_i - \mu_B)}{\sigma_A \sigma_B} \tag{4.2}$$

4.2 The impact of delay on the average performance of RC

Next, we proceed with the optimization of the delay, which is essentially the time window that we «feed» into the reservoir for training.

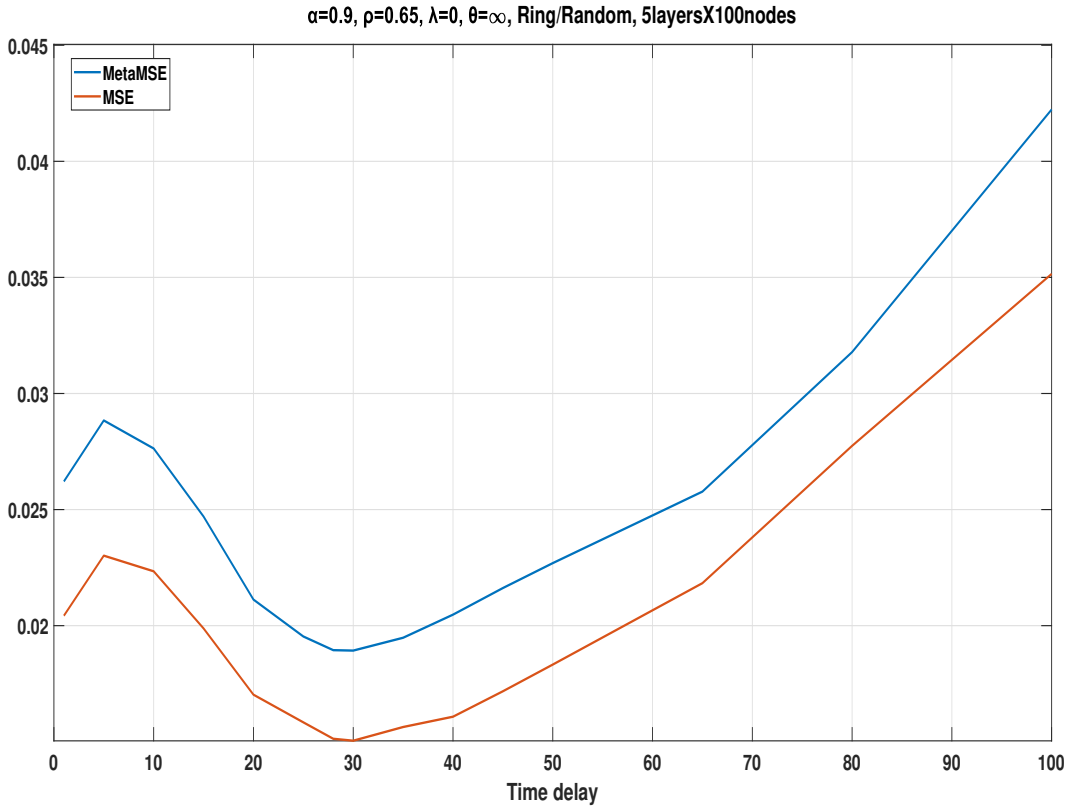


Figure 4.10

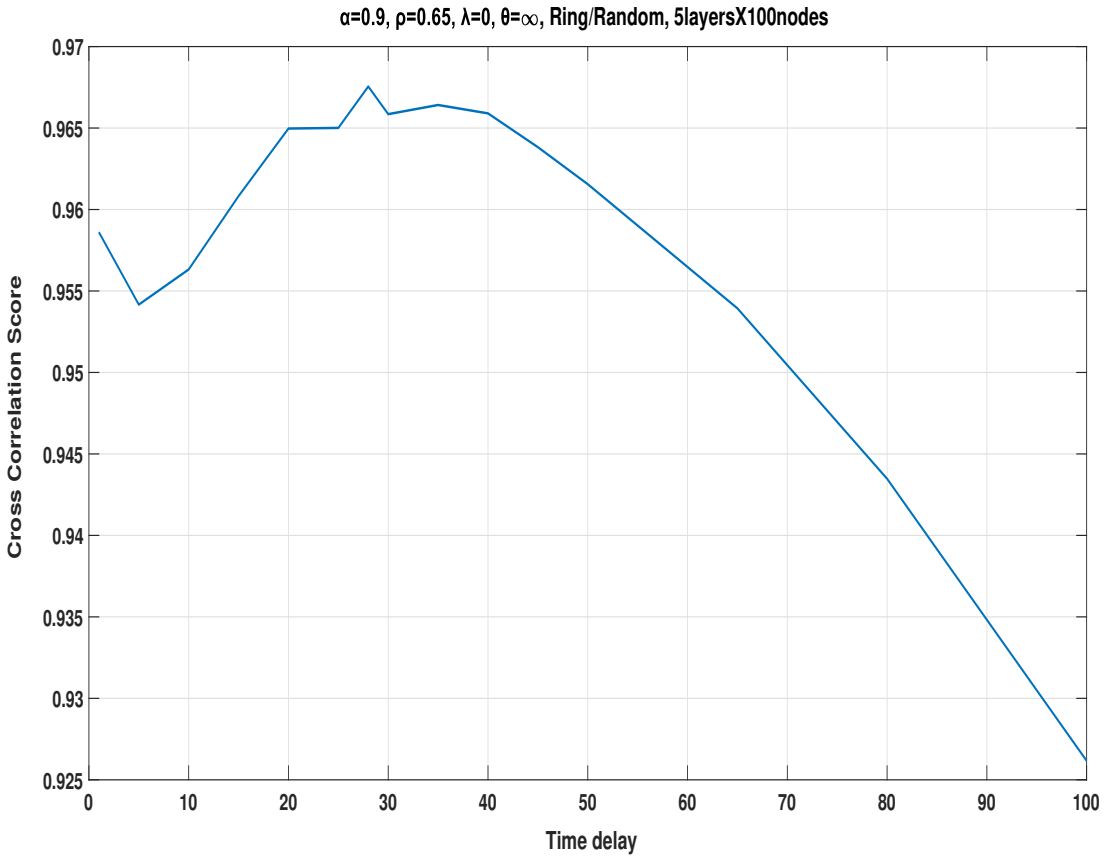


Figure 4.11

Above, we can see the evaluation of the impact of delay on the average performance of the reservoir using two different metrics: Cross-Correlation Score and Mean Squared Error, the latter in combination with MetaMSE. However, there is a common misconception that the larger the delay, the better the performance of the reservoir will be. This is completely debunked by the graphs, which indicate that there will always be an ideal time window, in our case somewhere around the past 28 to 30 time steps. Therefore, it appears that moving away from this time window in any direction tends to reduce the average performance.

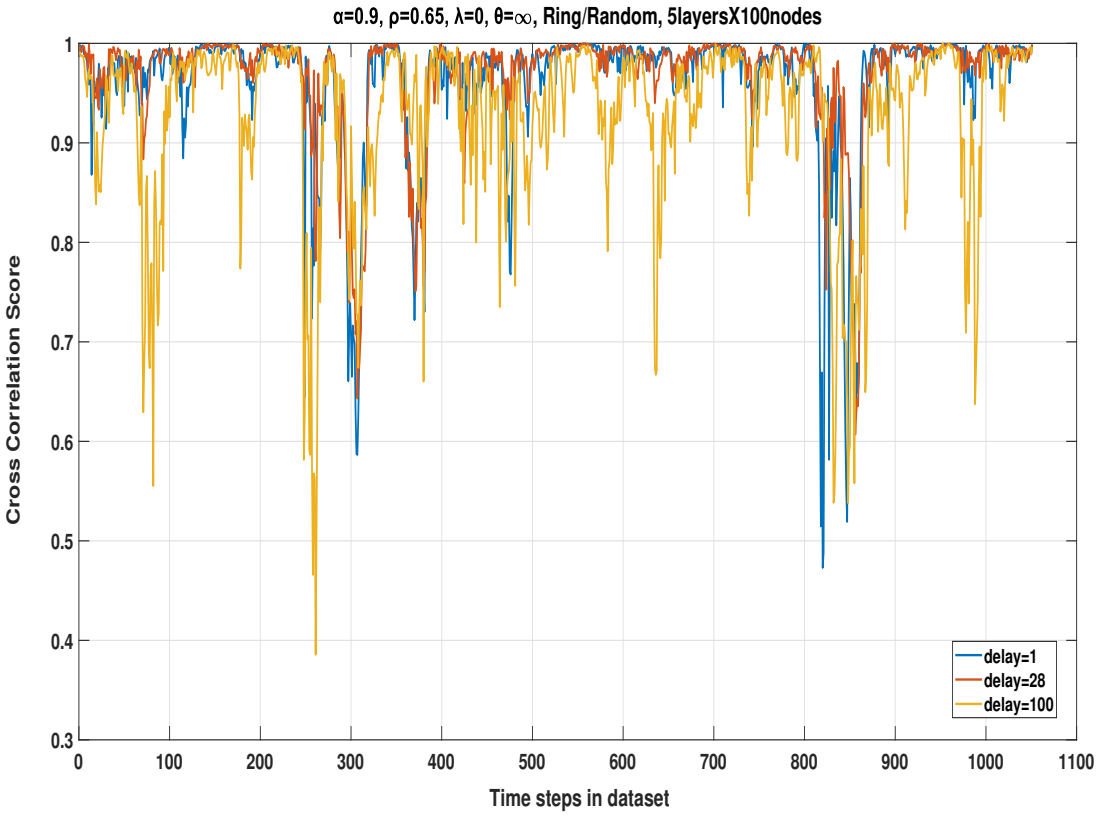
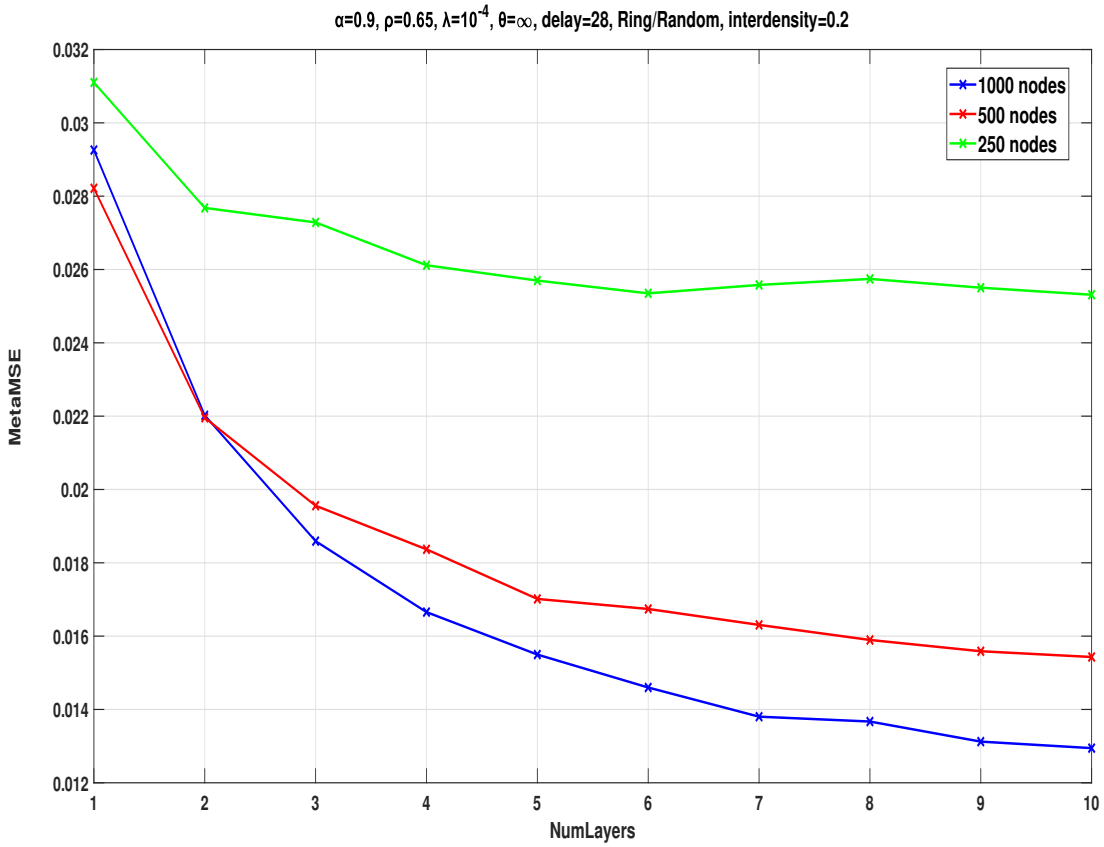


Figure 4.12

And finally, the last, but perhaps the most important graph, introduces a new way of assessing reservoirs or a new metric, which we call «*robustness*». The method described here is a very capable and powerful tool that allows us to look beyond average performances and in turn, gives us a complete picture at every individual moment. This way, «bad» days cannot be hidden by averaging.

4.3 The influence of the number of hidden layers on the reservoir's performance



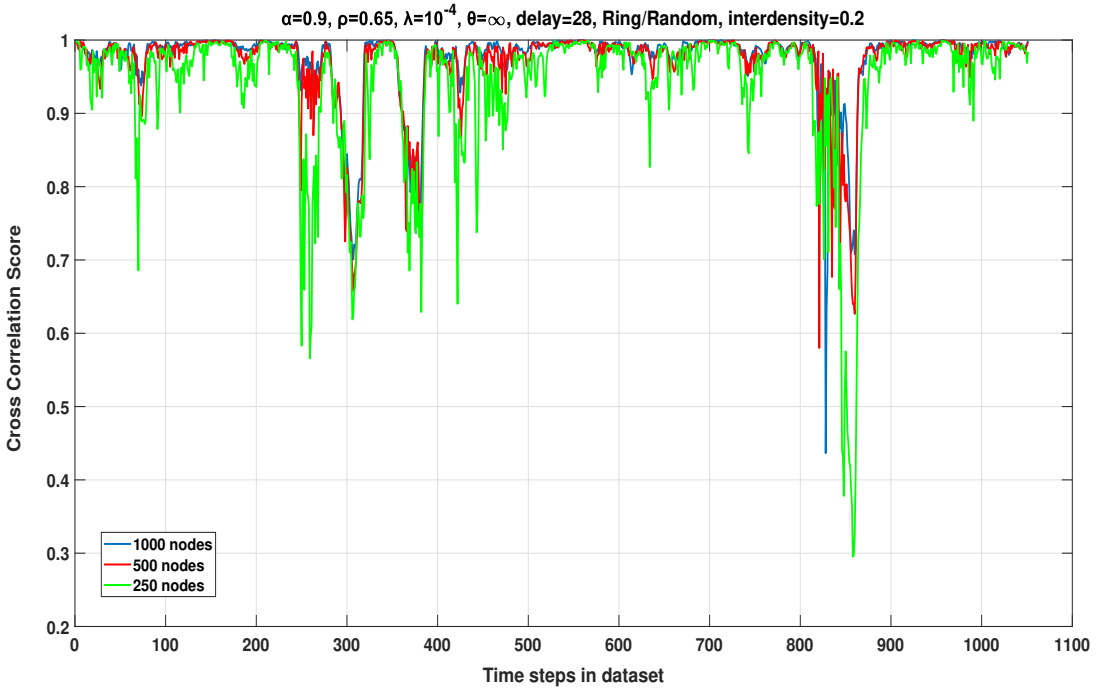
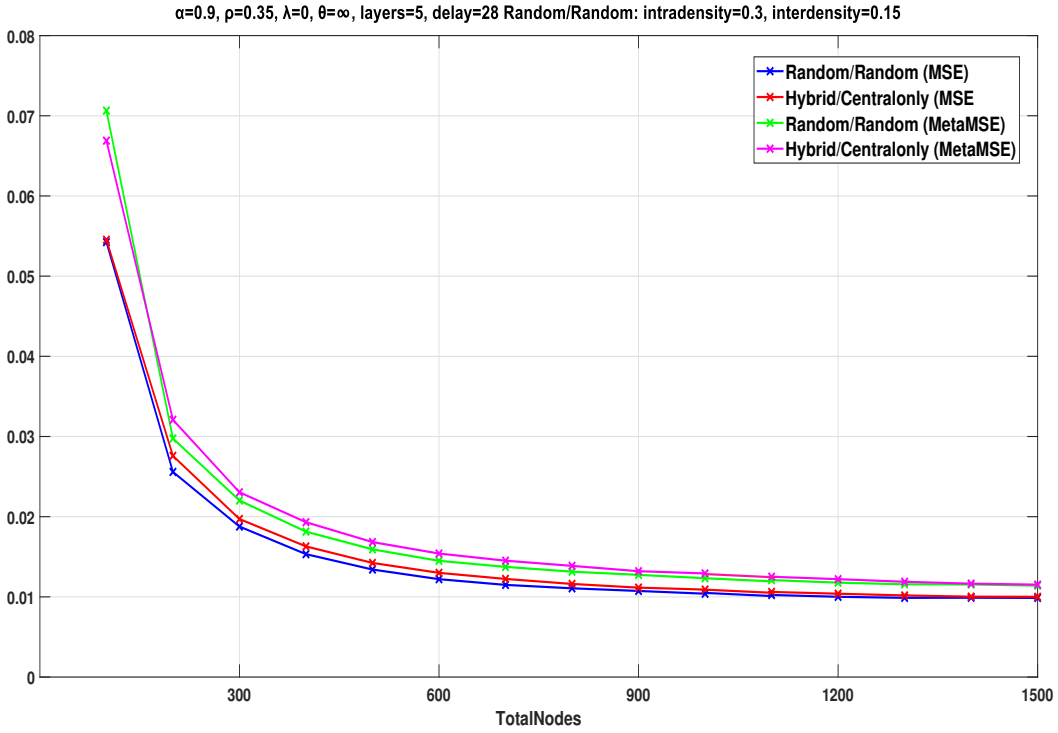


Figure 4.13

4.4 The effect of the total number of nodes and the reservoir's geometry



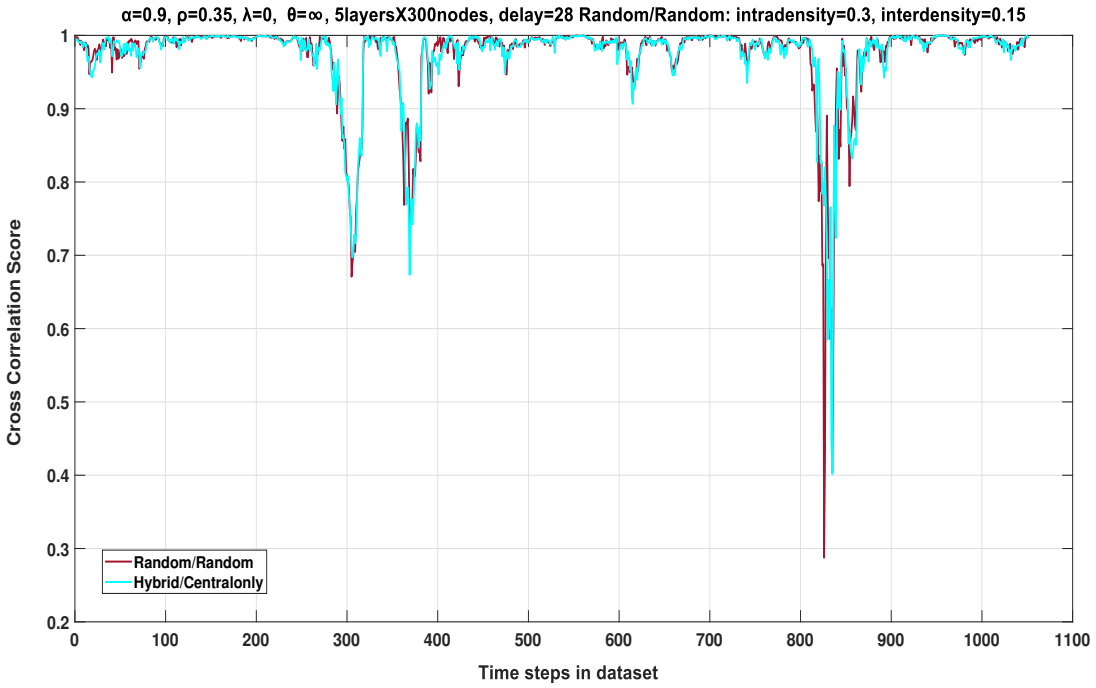
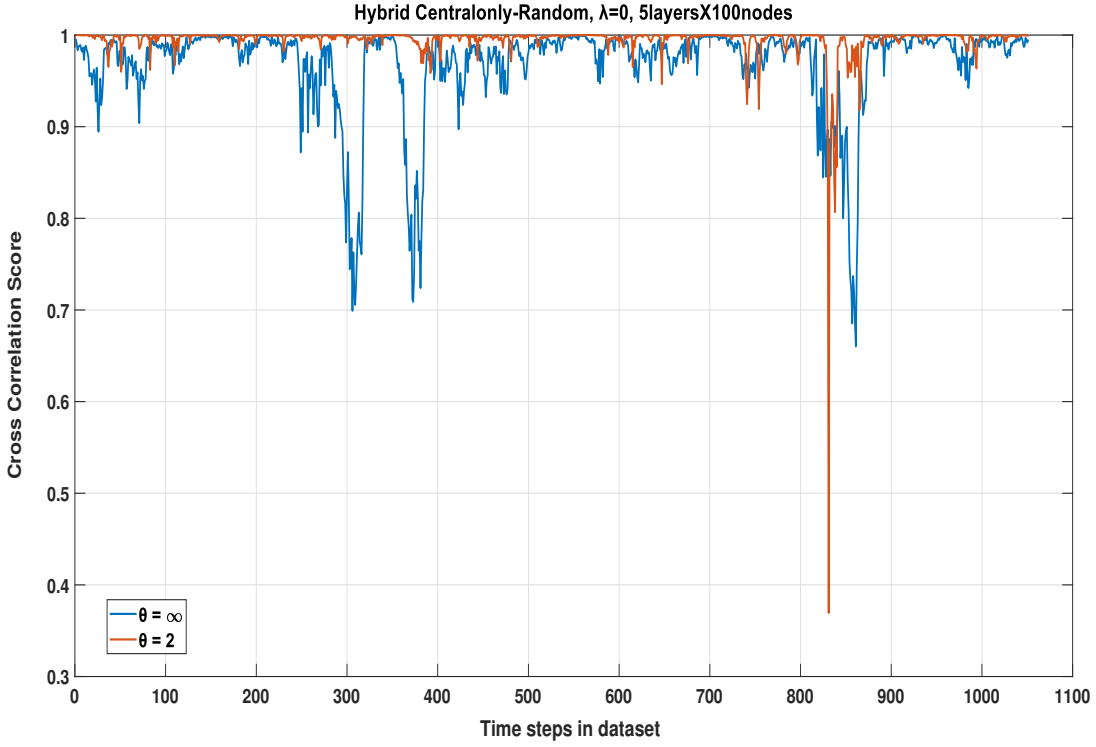
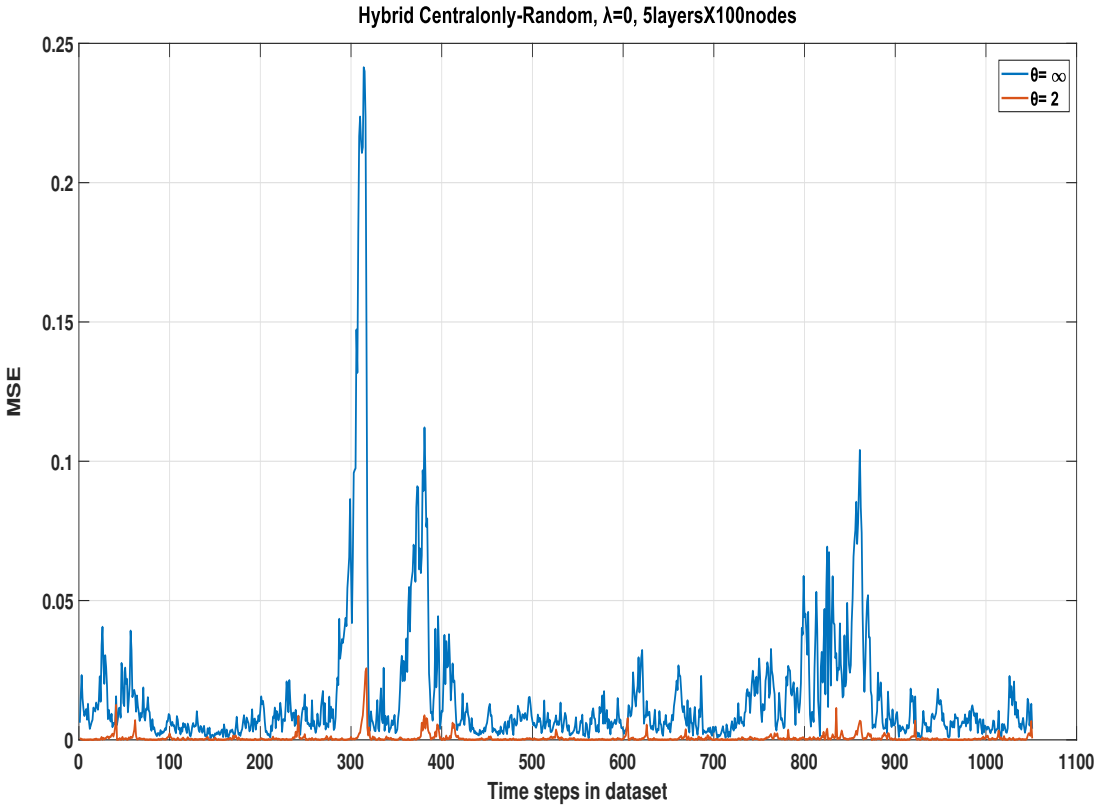


Figure 4.14

4.5 The effect of the problem's complexity on the robustness of the prediction




Hybrid Centralonly-Random, λ=0, 5layersX100nodes
θ= ∞
θ= 2
MSE
Time steps in dataset

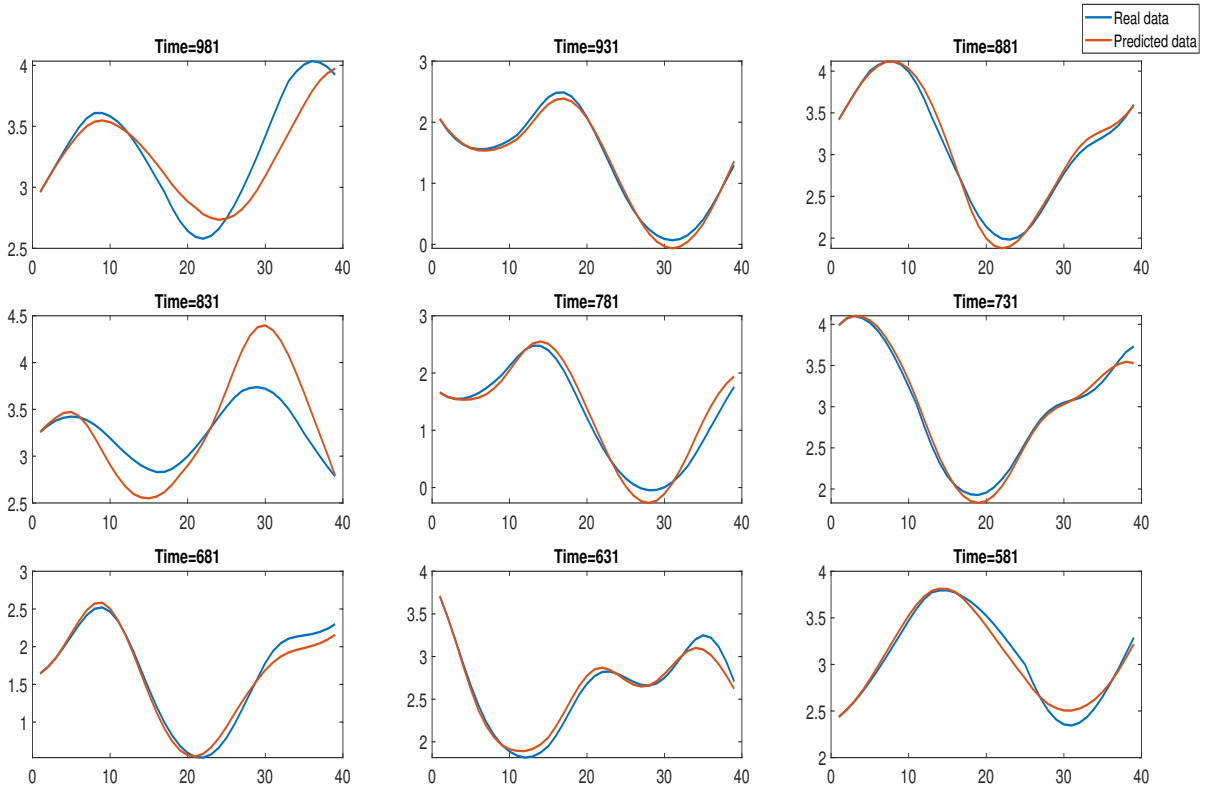

Real data
Predicted data
Time=981
Time=931
Time=881
Time=831
Time=781
Time=731
Time=681
Time=631
Time=581

Figure 4.15

Conclusions

- It is observed that classical metrics that evaluate the AVERAGE performance of the time series forecast (such as MSE, CCS) are not sufficient for the objective evaluation and optimization of the control parameters of the reservoir: They do not capture the ”bad moments.”
- There is a need to use new evaluation metrics that take into account the robustness of the predictions of the time series and not just the average performance.
- The ideal time window (delay) of the past that should be given to the reservoir at any given time is 28-30 moments (approximately twice the time of auto-correlation of the time series).
- In general, it was observed that both increasing the number of nodes and increasing the number of layers improves the average performance of the reservoir (until reaching a plateau).
- The Ring/Random topological arrangement was primarily observed as more efficient, and the centralonly implementation had a less satisfactory performance among the layers.

Future Work

- Application to real-world problems, such as meteorological, seismological data, and financial markets
- Preprocessing of input data (e.g., Fourier transformation)

Abbreviations

ESN: Echo State Networks

LSM: Liquid State Machines

TDR: Time Delay Reservoir

DFR: Delay Feedback Reservoir

RC: Reservoir Computing

RNN: Recursive Neural Network

MSE: Mean Squared Error

CCS: Cross Correlation Score

ANN: Artificial Neural Network

BP: Back Propagation

PRC: Physical Reservoir Computing

ERC: Electronic Reservoir Computing

ORC: Optical Reservoir Computing

LaNSCom: Laboratory of Non linear Systems and Complexity

Bibliography

[Ant23] Dr. Ioannis P. Antoniades. *MultiESN*. Version 1.3. unpublished software. 2022-2023.

[App+11] Lennert Appeltant et al. "Information processing using a single dynamical node as complex system". In: *Nature communications* 2.1 (2011), p. 468.

[Bol21] Erik Bollt. "On explaining the surprising success of reservoir computing forecaster of chaos? The universal machine learning dynamical system with contrast to VAR and DMD". In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 31.1 (2021).

[Bür+15] Jens Bürger et al. "Hierarchical composition of memristive networks for real-time computing". In: *Proceedings of the 2015 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH 15)*. IEEE. 2015, pp. 33–38.

[DSY11] Shifei Ding, Chunyang Su, and Junzhao Yu. "An optimizing BP neural network algorithm based on genetic algorithm". In: *Artificial intelligence review* 36 (2011), pp. 153–162.

[Guo+20] Tao Guo et al. "From memristive materials to neural networks". In: *ACS Applied Materials & Interfaces* 12.49 (2020), pp. 54243–54265.

[HM12] Hananel Hazan and Larry M. Manevitz. "Topological constraints and robustness in liquid state machines". In: *Expert Systems with Applications* 39.2 (2012), pp. 1597–1606. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2011.06.052. URL: https://www.sciencedirect.com/science/article/pii/S0957417411009523.

[HMS19] MP Hanias, L Magafas, and SG Stavrinides. ""Reverse Engineering" in Econophysics". In: *International Journal of Productivity Management and Assessment Technologies (IJPMAT)* 7.1 (2019), pp. 36–49.

[Jae12] Herbert Jaeger. *Long short-term memory in echo state networks: Details of a simulation study*. Tech. rep. Jacobs University Bremen, 2012.

[JLL22] Leisheng Jin, Zhuo Liu, and Lijie Li. "Chain-structure time-delay reservoir computing for synchronizing chaotic signal and an application to secure communication". In: *EURASIP Journal on Advances in Signal Processing* 2022.1 (2022), p. 68.

[Luk12] Mantas Lukoševičius. "A practical guide to applying echo state networks". In: *Neural Networks: Tricks of the Trade: Second Edition*. Springer, 2012, pp. 659–686.

[Nak20] Kohei Nakajima. “Physical reservoir computing—an introductory perspective”. In: *Japanese Journal of Applied Physics* 59.6 (2020), p. 060501.

[Noa91] Lyle Noakes. “The Takens embedding theorem”. In: *International Journal of Bifurcation and Chaos* 1.04 (1991), pp. 867–872.

[PSR19] Wachirawit Ponghiran, Gopalakrishnan Srinivasan, and Kaushik Roy. “Reinforcement learning with low-complexity liquid state machines”. In: *Frontiers in Neuroscience* 13 (2019), p. 883.

[Raf+20] Mushegh Rafayelyan et al. “Large-scale optical reservoir computing for spatiotemporal chaotic systems prediction”. In: *Physical Review X* 10.4 (2020), p. 041037.

[Sor+15] Miguel Soriano et al. “Minimal approach to neuro-inspired information processing”. In: *Frontiers in computational neuroscience* 9 (June 2015), p. 68. DOI: 10.3389/fncom.2015.00068.

[Su12] Jenny Su. *Reservoir Computing in Forecasting Financial Markets*. 2012.

[SVV07] Benjamin Schrauwen, David Verstraeten, and Jan Van Campenhout. “An overview of reservoir computing: theory, applications and implementations”. In: *Proceedings of the 15th european symposium on artificial neural networks. p. 471-482 2007*. 2007, pp. 471–482.

[Tri+10] Fabian Triefenbach et al. “Phoneme Recognition with Large Hierarchical Reservoirs”. In: *Advances in Neural Information Processing Systems*. Ed. by J. Lafferty et al. Vol. 23. Curran Associates, Inc., 2010. URL: https://proceedings.neurips.cc/paper_files/paper/2010/file/2ca65f58e35d9ad45bf7f3ae5cfd08f1-Paper.pdf.

[Tro+20a] Nathan Trouvain et al. “ReservoirPy: an Efficient and User-Friendly Library to Design Echo State Networks”. In: *ICANN 2020 - 29th International Conference on Artificial Neural Networks*. Bratislava, Slovakia, Sept. 2020. URL: https://inria.hal.science/hal-02595026.

[Tro+20b] Nathan Trouvain et al. “Reservoirpy: an efficient and user-friendly library to design echo state networks”. In: *International Conference on Artificial Neural Networks*. Springer. 2020, pp. 494–505.

[WG16] Gilles Wainrib and Mathieu N Galtier. “A local echo state property through the largest Lyapunov exponent”. In: *Neural Networks* 76 (2016), pp. 39–45.

[Wil09] Bogdan M Wilamowski. “Neural network architectures and learning algorithms”. In: *IEEE Industrial Electronics Magazine* 3.4 (2009), pp. 56–63.

[YJK12] Izzet B Yildiz, Herbert Jaeger, and Stefan J Kiebel. “Re-visiting the echo state property”. In: *Neural networks* 35 (2012), pp. 1–9.