University of Sheffield
Department of Automatic Control and Systems Engineering

Novel Reservoir Computing architectures and learning algorithms

A thesis submitted to the University of Sheffield for the degree of Doctor of Philosophy

Supervisors:
Prof. Daniel Coca, Dr. Dana Damian

Student:
Codrin-Alexandru Lupașcu

To my family,

Acknowledgements

I would like to thank my supervisors Daniel Coca and Dana Damian for being by my side during my years as a PhD Student. Moreover, I would like to thank Daniel for the Saturdays we have spent together playing table tennis and having interesting and passionate discussions. I would also like to thank my parents for their compassion, support and for cheering me up during difficult times. Many thanks go to Miruna Verdes and Cristian Genes for the very useful discussions that have inspired me to overcome theoretical and experimental barriers encountered in my research.

I thank all my colleagues from the Automatic Control and System Engineering department for their support, tips and reassuring discussions.

Many thanks go to my friends from Sheffield Dan and Simona Oara, Andrei Braitor, Cristian and Denisa Genes and Miruna Verdes for the awesome moments spent together playing boardgames, relaxing at the cinema, and for the recreational nature walking through Peak District.

I would like to thank my friends Andrei Balici, Bogdan Tarcau and Robert Onesim for being by my side when I first moved to England, making my transition to a new country much easier.

Abstract

Reservoir Computing is a framework for machine-learning that is derived from recurrent neural networks and involves only training the output map of a randomly initialized high dimensional neural network called reservoir.

This thesis addresses three significant gaps in the field of computational neuroscience, reservoir computing architectures and training algorithms.

First, it is shown that the original proof of the Universal approximation property of Reservoir Computing architectures does not hold, with regards to the lack of point separation of the output layer. Specifically, based on the Stone-Weierstrass theorem, it has been proven that any n-dimensional randomly generated Recurrent Neural Network could approximate any n-dimensional time-invariant system arbitrarily close only by training the output layer. In practice, several early numerical experiments raised the concern that this might not be the case as the approximation results seemed to be lower bounded. Moreover, some papers suggest that replacing the output-feedback connection of the reservoir with a state-feedback connection improves the approximation capabilities of the Reservoir Computing architecture.

This thesis addresses this issue by conducting a more thorough theoretical analysis which shows that indeed, the initial conditions set by the Stone-Weierstrass theorem which is used to prove the Universal Approximation property of Reservoir Computing architectures are not met.

Secondly, a new type of architecture and training strategy is proposed to ensure closed-loop stability and enhanced approximation capabilities for Echo State Networks. Current output-feedback Echo State Network architectures do not guarantee closed-loop stability when training the readout module, especially when the initial training errors are large. Moreover, the existence of the fixed output-feedback loop overloads the trainable readout module with two separate tasks that converge to different solutions: dynamical fitting of the reservoir with regards to the dynamics of the target system and the output fitting of the target signal.

This thesis develops two new training algorithms for the Echo State Networks with state-feedback gains. By using an Extended Kalmann Filter estimation method, training the input layer and the state-feedback gain decouples the two aforementioned problems. The indirect modification of the reservoirs dynamics via the state-feedback loop follows the rules of Reservoir

Computing (i.e., by not training the connections between the neurons inside the reservoir) and, at the same time, provides an enhanced level of accuracy compared to standard architectures and their associated training algorithms.

The first methodology fully trains the input layer and state-feedback gain via the Extended Kalmann Filter estimation method. After the dynamics of the closed-loop structure are set, an Orthogonal Forward Regression method is used to tune the readout by selecting the most relevant neurons and combination of neurons, in the case of non-linear readouts, to ensure maximum accuracy whilst avoiding the over-tuning problems associated with fully training all of the readouts components.

The second algorithm identifies the most relevant neurons that dictate the dynamics of the reservoir with regards to the dynamics of the target task via an Orthogonal Forward Selection algorithm. A reduced order input layer and state-feedback gain are then trained using the Extended Kalmann Filter estimation algorithm. The aim is to reduce the number of trainable parameters whilst keeping similar approximation accuracy to the previously mentioned algorithm. The readout is similarly trained using the same principles as in the first algorithm.

Thirdly, a new Echo State Network architecture is designed for computing several tasks given a randomly initialized reservoir with trainable input layers and state-feedback connections. State-of-the-art architectures use an overly dimensioned reservoir to adapt to the individual dynamics of each task whilst connecting several readouts in parallel. Because of the newly introduced trainable internal structures, in the form of a state-feedback gain and input layer, this approach becomes unfeasible as these structures alter the dynamics of the reservoir and cannot be used in a parallel fashion.

In this setting, this thesis proposes a novel switching state-feedback Echo State Network architecture. This new architecture enables processing multiple tasks, reduces the dimensionality of the reservoir and provides good approximation performance.

Contents

List of Figures

List of Tables

Nomenclature

A list of the variables and notations used in this thesis is defined below. All conventions set here will be observed throughout the proposed work, unless it is stated otherwise. Acronyms are presented after nomenclature.

$1_{[0,1]}$ - characteristic function

$[0,1]$ - closed interval zero to one

I_n - n-dimensional identity matrix

$\circ$ - function composition

$||\cdot||$ - second order norm

$||\cdot||_p$ - p-order norm

$<\cdot>$ - inner product

i - unit imaginary number

A^T - transpose of matrix A

A^{-1} - inverse of matrix A

$\mathbb{N}$ - set of natural numbers

$\mathbb{Z}$ - set of integer numbers

$\mathbb{R}$ - set of real numbers

$\mathbb{R}_+$ - set of positive real numbers

$\mathbb{R}_+^*$ - set of positive non-zero real numbers

$\mathbb{C}$ - set of complex numbers

$\{t_k\}_{k=1}^P$ - spike with timing sequence from $k = 1$ to P and amplitude 1

$\{a_k, t_k\}_{k=1}^P$ - spike with timing sequence and amplitude a_k from $k = 1$ to P

$L^1(\mathbb{R}_+^n)$ - Lebesque space of integrable n-dimensional functions on $\mathbb{R}_+^n$

$L^2(\mathbb{R})$ - Lebesque space of square integrable functions on $\mathbb{R}$

$C(\mathbb{R})$ - space of continuous functions over $\mathbb{R}$

$\mathscr{C}_R(E)$ - space of real continuous functions over the compact metric space E

Acronyms

AI - Artificial Intelligence

aFORCE - alternative First-Order, Reduced and Controlled Error

ANN - Artificial Neural Network

APRL - Atyia-Parlos Recurrent learning

BPDC - BackPropagation - DeCorrelation

BPTT - BackPropagation Through Time

EKF - Extended Kalmann Filter

ESN - Echo State Network

ERR - Error Reduction Ratio

ELM - Extreme Learning Machine

FHN - FitzHugh - Nagumo

FNN - Feedforward Neural Network

FORCE - First-Order, Reduced and Controlled Error

GP - Gaussian Process

HH - Hodgkin-Huxley

IAF - Integrate-and-Fire

LIF - Leaky Integrate-and-Fire

LS - Least Squares

LSM - Liquid State Machine

ML - Morris-Lecar

NARMAX - Non-linear AutoRegressive Moving Average with eXogeneous inputs

NG-RC - Next Generation Reservoir Computing

NN - Neural Network

NMSE - Normalised Mean Squared Error

OFR - Orthogonal Forward Regression

OFS - Orthogonal Forward Selection

OFRST - Orthogonal Forward Regression with Spike Trains

PBNP - Poisson-Boltzmann-Nernst-Planck

PNP - Poisson-Nernst-Planck

RK - Reproducing Kernel

RLS - Recursive Least Squares

RNN - Recurrent Neural Network

SESN - Switching Echo State Network

SISO - Single-Input-Single-Output

SNN - Spiking Neural Network

SNR - Signal-to-Noise Ratio

SLFN - Single Hidden Layer Feedforward Network

SQLF - Switched Quadratic Lyapunov Function

STDP - Spike-Timing-Dependent-Plasticity

Chapter 1

Introduction

1.1 Background and Motivation

Artificial Intelligence refers to systems that resemble human intelligence with the aim of performing certain tasks and improving iteratively based on their past experiences. Some important applications include: in healthcare for early detection of COVID-19 infection [1], education for developing intelligent tutoring systems [2] and in defense for developing cybernetic defense systems in banking [3]. Furthermore, in the era of big data, when one hour of YouTube videos is uploaded every second [4], automated methods for data analysis and decision making are paramount. In this context, Artificial Intelligence based solutions that exploit the adaptability and computing power of neural networks have been developed and deployed to identify patterns, predict future data and perform decision making under uncertainty [4].

The key element of any Artificial Intelligence based solution is the neural network that supports the learning and thinking processes. The term neural networks describes networks of biological neurons that constitute the nervous systems of animals. More recently, the term is used for a technology of parallel computation in which each element is an "artificial neuron" that is modeled based on its biological counterpart [5].

One of the main advantages of Artificial Intelligence/ Machine Learning solutions is that they can perform tasks without being explicitly programmed to do so. This is enabled by the fact that

these kind of approaches identify patterns in the data provided and then apply these patterns on the new data that is coming in. Traditional problem solving in computer science requires algorithms that tell the machine how to execute all steps in order to find the solution. Hence, no learning is involved in this process. In contrast, recent Artificial Intelligence based solutions are more effective because they help the machine develop its own algorithm. This is particularly useful for more advanced tasks in which manually creating the needed algorithms takes a long time [6].

The key elements of any machine learning-based solution are: the model training, validation, testing and deployment [6]. The most important step is training the model, as it dictates the performance of the overall model and ultimately, how well it will perform for the end-users. As part of the training process, the developer needs to take important decisions with regards to the training data to be used, the neural network architecture and the training algorithm. These decisions depend on factors such as algorithm-model complexity, interpretability requirements, computational power available and training time constraints.

The Artificial Neural Network (ANN) approach has been shown to be a general scheme for nonlinear dynamic system modelling and identification. Several Feedforward Neural Network (FNN) architectures have been considered in [7] and it has been demonstrated that FNNs have good capabilities for representing complex nonlinear systems. The multi-layer perceptron proves to be a universal model for non-linear systems. Unfortunately the error surface of a multi-layer perceptron model is often highly complex. The radial basis function network provides an alternative two-layer network structure. Each node in the hidden layer has a radially symmetric response around the node centre and linear learning laws can be derived. The main drawback, however, compared to the following two architectures, is that of computational complexity, as all the parameters of ANNs need to be trained.

Reservoir Computing is a machine-learning paradigm [8],[9] that consists of a neural network, in which only certain easy-to-train parameters are subject to modifications based on new tasks that are learned. This type of approach facilitates the use of easier to apply and more time-efficient training algorithms whilst having similar performances to that of fully trained neural networks.

Extreme learning machine (ELM) has been proposed for training single hidden layer FNNs (SLFNs). In ELM, the hidden nodes are randomly initiated and fixed. The only parameters susceptible to learning are the connections between the hidden layer and the output layer. In this way, ELM is formulated as a linear-in-the-parameter model [10]. Hyper-dimensional projection is a powerful computational tool itself and is used by ELMs in a manner similar to RCs reservoir layer but without the short- term memory component [11], which is unrealistic from a biological perspective.

This thesis addresses a number of dynamical fitting problems relevant to the field of Reservoir Computing [8],[9],[12],[13],[14], developing new type of training algorithms for the state-feedback

gain Echo State Network architecture [14] as well as developing a new type of Echo State Network architecture for processing multiple tasks using the same reservoir.

One of the significant breakthroughs in the domain of Reservoir Computing is the proof that this type of architecture is an Universal Approximator [15]. The main theorem in [15] relies on the Stone-Weierstrass theorem of approximation and states that any given n-dimensional reservoir could approximate arbitrarily close any given n-dimensional time-invariant target system with fading memory. In practice, however, several numerical studies [8],[16],[17] and theoretical analysis papers [13],[14] have shown that the current output-feedback Reservoir Computing architectures could be improved to further enhance their approximation capabilities, by introducing a state-feedback loop [14]. Moreover, numerical simulations show that current state-of-the-art algorithms and architectures are lower bounded. However, these theoretical and numerical observations contradict the Universal Approximation property of the original architecture.

This thesis covers an important theoretical advancement that analyses the theorems and results on the approximation capabilities of current Reservoir Computing structures [8],[9],[12]. The conditions that need to be met in order to apply the Stone-Weierstrass theorem of approximation, used in the original proof [8], are not actually met.

An important problem for Echo State Networks is to develop mathematical formulations for training the readout in different types of neural architectures, as well as fine-tune the initialization hyper parameters [12],[16],[18],[19].

This problem was first addressed by Jaeger [9], who demonstrated that clamping the fixed output-feedback loop with the target signal, instead of using the networks output during training, would not lead to an unstable closed-loop system if the initial training errors between the original target and the networks output were small. The major disadvantage, however, comes when these initial errors are large, which usually lead to an unstable closed-loop system.

Sussillo [12] developed a new training strategy that removed the need for clamping the output-feedback and relies on a recursive and efficient method that keeps the training error small whilst making an impactful output weight modification. However, both strategies and derived algorithms suffer from the problem mentioned above: the readout module is overloaded with solving two problems that competing solutions: dynamical fitting of the reservoir via the output-feedback gain loop and output fitting of the target signal. The field would benefit from a new training strategy that would overcome the limitations imposed by using the readout, through a fixed and randomly initialized output-feedback gain, as a mean of modifying the dynamics of the fixed reservoir at the same time as fitting the output target signal.

To address this, a new architecture is introduced for improving the dynamical approximation of Reservoir Computing systems represented in the form of trainable input layer - state-feedback gain Echo State Networks. Based on this new architecture, new training algorithms are introduced

for alternatively training the input layer - state-feedback gain to ensure that the dynamics of the reservoir are fine-tuned to the dynamics of the target system, whilst training the readout for the sole purpose of fitting the output of the network to the target signal. Two different methods of training the input layer - state-feedback gain pairs via an Extended Kalmann Filter estimation method [20],[21],[22] are introduced: a full training algorithm that modifies the behaviour of each individual neuron inside the reservoir via their connections to the input layer and feedback gain, and a reduced order method, where only the most relevant neurons are selected to approximate the dominant dynamics of the target system. Then, the output layer is trained via an Orthogonal Forward Regression algorithm [23],[24] that selects only the most important neuron connections in order to avoid some over-fitting problems associated with fully training all the parameters whilst keeping maximum approximation accuracy. The proposed approaches offer significantly faster and more accurate results than the state-of-the-art FORCE method [12] and NG-RC algorithm [19].

One advantage of current generation Reservoir Computing architectures is that they are able to perform multiple tasks in parallel, by coupling at the same time individual trainable readout modules for each corresponding task to the randomly initialized reservoir. Due to the inherited dynamical boundaries of this type of structure, a larger reservoir is required to obtain satisfactory approximation results for each separate task. The dimensionality of the network is inefficiently chosen as a trade-off to assure that the dynamics cover a larger frequency spectrum, that would theoretically cover some of the dynamics of the unknown target tasks.

The field would benefit from a new architecture that would reduce the dimensionality of the reservoir and would maximize the approximation capabilities of Echo State Networks with respect to performing several target tasks. By using the newly trainable input layer - state-feedback gain Echo State Network architecture and its proposed training methodologies, a novel switching architecture is developed. Instead of using a parallel architecture that is over-dimensioned and not specifically designed to the dynamics of the target tasks, it would be more efficient, memory-wise to develop a smaller reservoir that would benefit from the trainable input layer - state-feedback gain pairs that fine-tune the dynamics of the closed-loop reservoir for each individual task which, in turn, would offer better approximation results. Necessary stability conditions are presented to guarantee switching stability between the subsystems. The proposed approach offers significant improvements with regards to the approximation capability for each task at the cost of higher free-running times compared to the standard parallel architecture.

1.2 Overview of the thesis

This section presents a brief overview for each chapter of this thesis.

- Chapter 2 gives an overview of the literature with regards to Reservoir Computing, with a particular focus on Liquid State Machines and Echo State Networks. For each, generic architectures and training algorithms are presented and the shortcomings of the most popular approaches are discussed.

- Chapter 3 gives an overview of the theoretical framework of Liquid State Machines. The chapter begins with a brief review of the fundamental theory behind operators. The existing results regarding the Universal Approximation property and the Stone-Weierstrass theorem that were presented in Chapter 2 are further discussed in this chapter by introducing appropriate theoretical results which indicate some limitations regarding the approximation capabilities of fixed output-feedback structures that were not considered throughout the literature. The initial conditions of applying the Stone-Weierstrass theorem to prove that current output-feedback Reservoir Computing architectures hold universal approximation are analysed and show that actually, they cannot be met. Emphasis is put on an experimental example that underpins the approximation boundaries, which are directly related to the dynamics of the reservoir with regards to the dynamics of the linear target system.

- Chapter 4 introduces a novel training algorithm for Echo State Networks. The chapter begins with the limitations introduced by the presence of a fixed output-feedback gain connection to the reservoir, with regards to the approximation capabilities of the closed-loop architecture. The algorithm reformulates the current architecture by changing the fixed output-feedback gain with a trainable input layer - state-feedback gain pair that drastically improves the approximation results when fitting the dynamics of the closed-loop structure to the dynamics of the target system while the trainable readout is only used for the sole purpose of output fitting the target signal. This new result forms the basis for two new training algorithms that are significantly more efficient than the state-of-the-art methods. Numerical studies were performed to show and isolate the advantages of the newly introduced algorithms with regards to the state-of-the-art training methods and to show their increased approximation performance.

- Chapter 5 builds on the theoretical highlights of the newly introduced training algorithm presented in Chapter 4 by introducing a novel training algorithm which selects a dimensional variable state-feedback gain and input layer based on the most relevant neurons that dictate the dynamics of the reservoir with regards to the dynamics of the target system. The main

advantage introduced by selecting the trainable parameters of the structure is that it avoids certain over-fitting problems that might be present in fully trained structures. The same numerical studies that were carried out in the previous chapter show similar, if not improved performances in some cases, between a fully trained input layer - state-feedback gain Echo State Network (algorithm presented in Chapter 4) and the reduced dimension version of the input layer - state-feedback gain Echo State Network introduced in this chapter.

- Chapter 6 introduces a new type of architecture, namely Switching Echo State Networks. State-of-the-art architectures perform multiple tasks on a single feed-forward reservoir by coupling several trainable readout modules. Based on the advantages of the newly introduced training algorithms presented in Chapter 4 and 5, having a flexible reservoir, which switches between the pairs of input layers and state-feedback gains to match the dynamics of each individual task, and switching between readouts to match the target signal, offers better numerical approximation results when compared to the state-of-the-art fixed output-feedback Echo State Network coupled with parallel readouts. Several closed-loop construction conditions are also analysed and introduced in order to guarantee the stability of the overall switching system. Experimental studies on data recorded from a fruitfly photoreceptor were performed to show an increase in performance of the newly proposed architecture when compared to previous NARMAX solutions.

- Chapter 7 contains concluding remarks and future work ideas.

Chapter 2

Overview of Reservoir Computing

2.1 Introduction

Recurrent Neural Networks (RNNs) are Artificial Neural Networks (ANNs) that incorporate one or more feedback connections [25] that can be local, from the output to the input of individual neurons, or global, from the outputs to the inputs of the network. The incorporation of feedback connections means that RNNs are in fact non-linear dynamical systems that can be used to model complex dynamical behavior [26],[27],[28], unlike feedforward ANNs that define a static mapping between inputs and outputs [29],[30]. Haykin [5] identifies three fundamentally different classes of networks: Single-Layer Feedforward Networks, where the input layer of neurons projects information to the output layer of nodes, but not vice-versa, MultiLayer Feedforward Networks, where, between the input and output layer of neurons, there exists one or more "hidden" layers of nodes (the information that is transmitted by the input layer is first projected through the hidden layers, in which a collection of neurons operate together at a specific depth inside the network, and from those layers it is transmitted to the output layer), and Recurrent Networks that distinguish themselves from MultiLayer Feedforward Networks by having at least one feedback loop that transmits information to one or more previous layers.

Several RNN architectures have been derived based on two main directions. First, according to the type of neuron activation function, RNNs can be classified into spiking [8],[16],[17],[31],[32] and non-spiking RNNs [9],[12],[19]. Secondly, according to the network parameters that are modified in training, we distinguish fully trained RNNs [12],[33],[34],[35],[36], where all the connections between neurons are constantly changed, and partially trained RNNs, where just a particular set of connections are modified (e.g., changing only the connections of the output layer, of the feedback loop, etc.) [8],[9],[12],[28],[32].

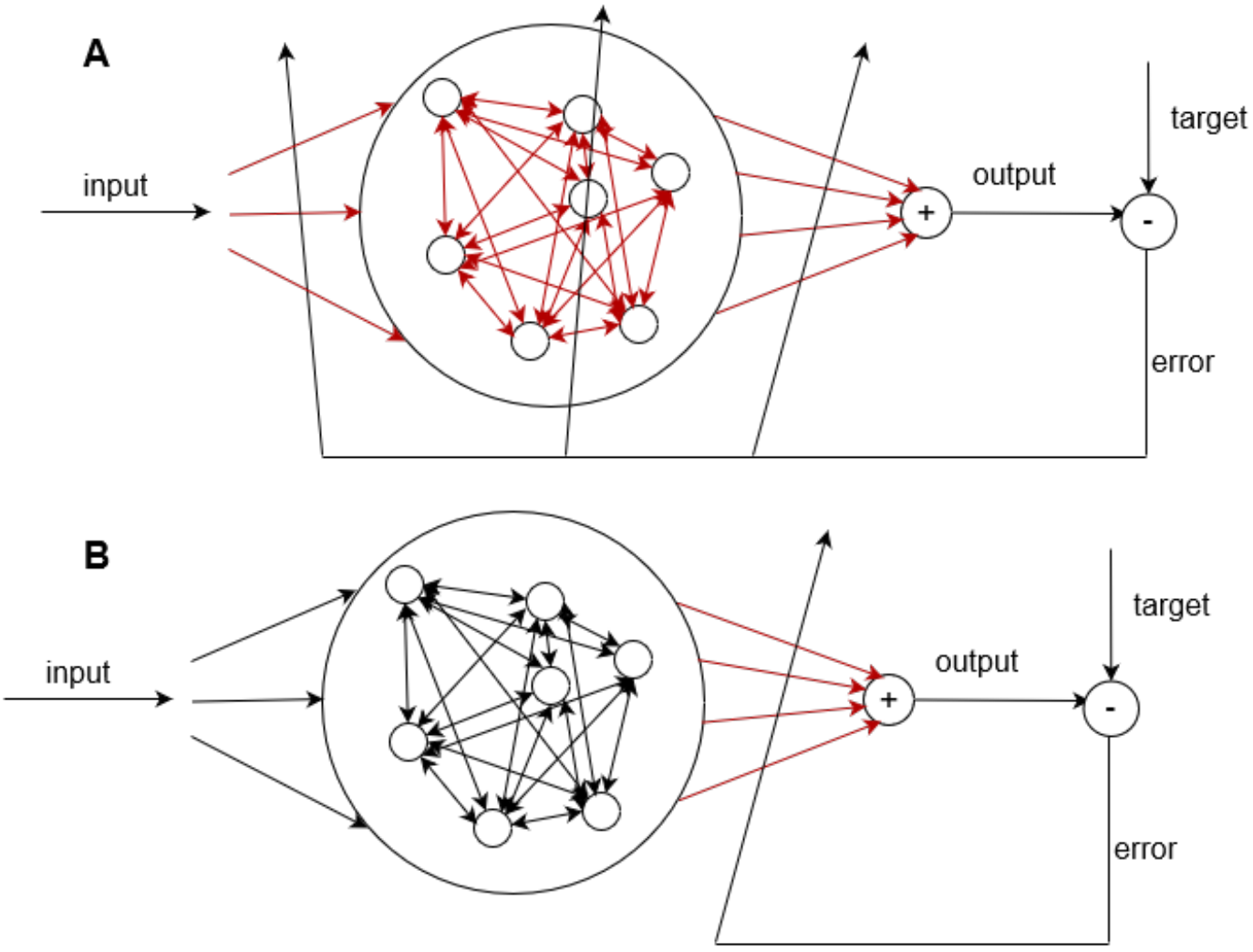

Figure 2.1: **Different RNN architectures based on the employed training strategies. The red arrows represent the parameters of the network that are trained:** A) Fully trained RNN structure B) Partially trained RNN structure

Widely used fully connected continuous RNN training algorithms, that are based on the gradient descent method, such as: the Backpropagation Through Time (BPTT) algorithm [33], the Atiya-Parlos Recurrent Learning (APRL) method [34], Extended Kalmann Filters (EKF) estimation

[35], and Expectation Maximization algorithm [36], methods which minimize the error between the output of the network and the target output, by modifying each weight in proportion to the corresponding partial derivative of the error with respect to that weight. Atiya [34] mentions that this class of recurrent training algorithms can be classified into five categories: the forward propagation approach [37],[38], where the weights are updated on-line; the truncated Backpropagation Through Time [39],[40],[41], where the RNN is transformed into an equivalent FNN off-line; the fast forward propagation approach [42] which computes the gradients recursively, on-line; the Greens Function approach [43] that improves the computational complexity of the previous category, and the block update approach [44],[37],[45] which updates the weights by grouping the data points into blocks, leading to a decrease in computational complexity. However, the biggest issue of the first generation RNN architecture is that all the above training categories are more complex compared to the methodologies used to train feedforward network structures, due to the presence of local and global feedback loops. The difficulty in efficiently computing the gradient used to train the weights leads to complex and slow converging methods [8],[15],[16],[18],[34]. This is due to the fact that the on-line methods operate at a higher complexity of $O(N^3)$ and $O(N^4)$ making this class difficult to scale to larger networks, whilst, the most efficient off-line method, the truncated Backpropagation Through Time, operates at a lower complexity of $O(N^2)$, but is not practical for on-line applications as the gradients have to be propagated to the initial time for each new data point.

If only the global feedback is present, from the output to the input, the RNN could be trained using algorithms developed in the control space, such as the EKF [20],[21],[22] for the feedback loop and input layer identification. This type of method has never been fully explored in this space of RNNs [14], as Maass only suggests that training a state-feedback loop could improve the approximation capabilities of RNNs at a fraction of the cost of fully training all the connections of RNNs. Sussillo in [12] proposes a state-feedback loop gain represented in the form of a smaller network. Only the input layer of the newly introduced structure is trained, which improves the approximation results, but not in a substantial way. Other papers explored fully training the RNN using the EKF with good results [46],[47]. Throughout this thesis, we will pick up on these methods and follow up on the suggestions made by Maass [14] by using them efficiently in newly proposed training algorithms for Echo State Networks. We will also comment on the architecture proposed by Sussillo, in the sense that, training the parameters of the input layer of the feedback network will not affect, in a qualitative manner, the approximation capabilities of the closed-loop structure.

In the direction of spiking RNNs, a series of gradient descent based supervised learning algorithms have been proposed: the SpikeProp through time [48],[49],[50], which optimizes the spiking RNNs dynamics on the timescale of individual spikes by introducing a differentiable formulation of the spiking RNNs to derive the exact gradients calculation and the Kullback-Leibler divergence algorithm [51],[52],[53] which minimizes the divergence between the distribution of the

desired output spike trains and the spike trains produced by the network. Other algorithms with different biologically plausible rules have been proposed, such as [54],[55],[56]. The main problem with fully training spiking RNNs using these types of methods is that the gradient descent methods do not guarantee the convergence of the solution, as the information used in the construction of the gradients degenerates and might become ill-defined, due to the presence of the bifurcations when choosing the path to the optimal solutions. Moreover, parameter updates are computationally expensive, resulting in long training times, making it feasible only for networks with neurons in the order of tens of units [18]. Thus, new developments had to be made in order to solve the previously mentioned issues, either by creating more efficient training algorithms or by changing the philosophy of the networks architecture design.

The following sections within this chapter will cover generic RC architectures, properties and training methodologies, neuron models used in LSMs, LSM architectures, properties and training algorithms, typical neuron activation functions for ESNs, ESN architectures, properties and training algorithms accompanied by some concluding remarks regarding the advancements and gaps in the literature.

2.2 Reservoir Computing

2.2.1 Generic architecture and training algorithms

Reservoir Computing (RC) is a machine-learning paradigm that is based on the standard state-space model RNN architecture and involves only training the output map of a randomly initialized high dimensional dynamical reservoir [8],[9],[31]. A typical RC machine consists of an input layer that passes the input signal to a fixed, randomly initialized dynamical system, in the form of an NN, and a readout that performs a transformation on the systems states. Usually, the dynamical system is chosen as a non-linear transformation of the input and the readout is typically chosen as a linear combination of the RCs internal states. Figure 2.1 B) depicts a typical RC architecture.

The state space model of a generic RNN:

$$\dot{x}(t) = f(x(t), u(t), W) \tag{2.1}$$

with the output equation

$$y(t) = \mathbf{g}(x(t)) = w_{out-1} g_1(x) + w_{out-2} g_2(x) + ... + w_{out-p} g_p(x) \tag{2.2}$$

Each neuron $i, i \leq n$ from (2.1) can be described separately:

$$\dot{x}_i(t) = f_i(x_i(t), u(t)) \tag{2.3}$$

where $u : \mathbb{R} \to \mathbb{R}^m$ is the input injected in the network, $x : \mathbb{R}^n \to \mathbb{R}^n$ are the states of the network, $y : \mathbb{R}^n \to \mathbb{R}^p$ is the output, $f : \mathbb{R}^n \to \mathbb{R}^n, f = [f_1, ..., f_n]^T$ is the network of neuron activation functions, usually with $f_1 = ... = f_n$, $W \in \mathbb{R}^{n \times n}$ is the connection matrix between neurons, $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^m$ represents the linear-in-the-parameter output mapping function of x, also referred to as the readout, $w_{out-1}, ..., w_{out-p}$ are the output weights and $g_1, ..., g_p : \mathbb{R}^n \to \mathbb{R}^p$. The most common choices for the readout modules mapping function are represented by memory-less functions, that are not required to retain any memory of previous states, and are linear $\mathbf{g}(x(t)) = W_{out} x(t), W_{out} \in \mathbb{R}^{p \times n}$ [8],[9],[32] or non-linear (i.e., element-wise applied sigmoid, polynomial) when dealing with highly non-linear tasks [18].

The activation function of a neuron is a function that introduces non-linearities into the output of that neuron given an input or set of inputs. The activation functions can be classified into two main categories: continuous functions (e.g., linear, sigmoid functions, etc.) and spiking functions

(e.g., Hodgkin-Huxley, Integrate-and-Fire, etc.). The choice of activation function has a large impact on the capability and performance of the NN. Typically, the first category of functions are differentiable, as the main method of training an NN is based on the backpropagation of errors [39] that requires first-order derivation. The advantage of using spiking functions is that the associated computational power costs are lower, compared to continuous signals whilst transmitting more information through the exact spiking time or through the inter-spike time interval. However, the main issue of fully training the connections of spiking RNNs is that the existing training methods were developed for continuous (non-spiking) RNNs and the adapted methods are only reliable for small networks (in the order of tens of neurons) [18] due to increased training times.

Existing artificial networks could not offer a numerical tractable problem in the representation of highly connected recurrent circuits present in the neocortical column of the biological brain [57] as it required ANNs to possess an immense number of states for processing and storing the high-varying inputs, which made existing training algorithms time inefficient [8]. Thus, real-time computing of time-varying inputs would be inspired by nature, namely, the liquid [58]. It only has one stable state, that is, the resting state. Any other states are generated by the presence of an input, considered as a perturbation to the system and are memorized by the liquids recurrent local and global feedback loops.

A number of theoretical results regarding the approximation power and performance of reservoir computing models have been formulated using dynamical systems theory [8],[14]. These results will be reviewed in the following sections.

2.2.1.1 General learning framework and principles

The main training methodology that has been extensively used for conventional RC architectures with linear-in-the-parameters readouts in fitting the output map parameters accordingly is represented by the Least Squares (LS) algorithm [59],[60]. Other linear-in-the-parameters optimization algorithms, such as the Orthogonal Forward Regression (OFR) algorithm [23],[32], finds the minimum number of parameters that should be trained to avoid over-fitting problems and maintain maximum approximation performance by minimizing the following cost function:

$$E = ||y - y_{target}||^2 \tag{2.4}$$

where y_{target} is the target output signal.

Other approaches in improving the performance of existing training methods have been mentioned in [18]. Finding a way to design and initialize the reservoir for a specific task would dramatically improve the readout training time and reduce approximation errors as the dynamics

of the liquid would be specifically designed for the task at hand. Moreover, having more neurons in the network that are sparsely connected (establishing up to 20% of the total number of possible connections) produces a "rich" set of dynamics. Decreasing the spectral radius of the connection matrix of the network ($\rho(W) = \max_{i \leq n}(\lambda_i)$, where λ_i is the i-th eigenvalue of W), could represent another solution in improving training results for particular tasks (i.e., when the target system is represented by a fast response linear discrete system with small spectral radius, the reservoir can be initialized with a spectral radius lesser than 1, which is the condition for stability in the case of discrete systems, in order to only capture fast dynamics). Other developments of reducing approximation errors suggested by the survey paper [18] include using supervised pre-training evolutionary algorithms for tuning the reservoir. Based on key parameters such as size, maximum eigenvalue, connection topology, etc., several candidate networks are generated and the best candidates, with regards to the approximation error, are chosen to be the parents of the next generations evolution. The new candidates are created by either mating two or more parent NNs by randomly combining the weights of the connectivity matrices or by self-mating, meaning that the parent becomes the new child. After the childs are created, they become subject to mutation, by adding or removing nodes, adding noise to the weights, changing the activation functions, etc. Such methods include the Kroenecker extension [61], which is a self multiplication evolutionary method, and the Evolino method [62], which, in the first instance, computes the readout for each candidate reservoir via standard regression and, in the second phase, selects the best reservoirs as the parents for the new generation and creates the childs by mating between them.

Several construction methods of the neurons functions are suggested in [18], such as: Support Vector Machines [63], Radial Basis Functions [64], Slow Feature Analysis [65] and Probability Mixture Models [66]. While usually the reservoir dynamics are non-linear and the readout is a linear map, an alternative reversed architecture with a linear dynamic reservoir and non-linear readout has also been taken into consideration [67],[68],[69]. When memory-less readouts are ineffective, coupling standard readouts with delays can solve the issue of learning the long-term dependencies present inside multi-time-scale tasks.

2.2.2 Universal approximation properties of Reservoir Computers

A machine possesses the universal approximation property [8] if any time-invariant filter F that has fading memory which maps input functions $u(t)$ to output functions $y(t)$ can be approximated to any degree of precision.

Definition 2.1 *Functions $F : C(\mathbb{R}) \to C(\mathbb{R})$ that map input functions $u(\cdot)$ on output functions $y(\cdot)$ are referred to as filters in neuroscience and engineering.*

Definition 2.2 *A filter $F : C(\mathbb{R}) \to C(\mathbb{R})$ is time-invariant if:*

$$U_\tau F u = F U_\tau u, \forall u \in \mathscr{U}, \tau \in \mathbb{R}$$

where $U_\tau u(t) \triangleq u(t-\tau)$ is the τ-delay operator.

Definition 2.3 *A filter $F : C(\mathbb{R}) \to C(\mathbb{R})$ has fading memory on a subset $\mathscr{U} \subset C(\mathbb{R})$ if there exists a decreasing function $\mu(t) : \mathbb{R}_+ \to (0,1], \lim_{t\to\infty} \mu(t) = 0$, such that $\forall u \in \mathscr{U}$ and $\varepsilon > 0$ there exists $\delta > 0$ such that $\forall v \in \mathscr{U}$ satisfies:*

$$\sup_{t\leq 0} |u(t) - v(t)| \mu(-t) < \delta \implies |(Fu)(0) - (Fv)(0)| < \varepsilon.$$

In particular, [70] define operators as having fading memory if two input signals which are relatively close from a temporal perspective, yield present outputs which are close.

Definition 2.4 A class *CB* of filters is a class of functions which has the point-wise separation property with regards to a class of input functions U, if for any two functions $u(\cdot), v(\cdot) \in U, u(s) \neq v(s), s \leqslant 0$ there exists $B \in CB$ such that $(Bu)(0) \neq (Bv)(0)$.

Theorem 2.1 from [15]. *Consider a space U^n of input functions where $U = \{u : \mathbb{R} \to [-B, B] : |u(t) - u(s)| \leqslant B' \cdot |t-s| \ \forall t, s \in \mathbb{R}\}$ for some $B, B' > 0$ (thus, U is a class of uniformly bounded and Lipschitz-continuous functions). Assume that CB is some arbitrary class of time-invariant filters with fading memory that has the point-wise separation property. Furthermore, assume that CF is some arbitrary class of functions that satisfies the approximation property.*

Then any given time-invariant filter F that has fading memory can be approximated by Liquid State Machines (LSMs) with liquid filters L^M composed from basis filters in CB and readout maps f^M chosen from CF. More precisely, for every $\varepsilon > 0$, there exist $m \in \mathbb{N}, B_1, ..., B_m \in CB$ and $f^M \in CF$ so that the output $y(\cdot)$ of the LSM M with liquid filter L^M composed of $B_1, ..., B_m$, that is, $(L^M u)(t) = \langle (B_1 u)(t), ..., (B_m u)(t) \rangle$, and readout map f^M satisfies for all $u(\cdot) \in U^n$ and all $t \in \mathbb{R}$ $||(Fu)(t) - y(t)|| \leqslant \varepsilon$.

Theorem 2.2 from [14]. *A large class S_n of systems of differential equations of the form*

$$x_i'(t) = f_i(x_1(t), ..., x_n(t)) + g_i(x_1(t), ..., x_n(t)) v(t), i = 1, ..., n \tag{2.5}$$

are in the following sense universal for analog computing:

This system (2.5) can respond to an external input $u(t)$ with the dynamics of any n^t order differential equation of the form

$$z^{(n)}(t) = G(z(t), z'(t), ..., z^{(n-1)}(t)) + u(t) \tag{2.6}$$

(for arbitrary smooth functions $G : \mathbb{R}^n \to \mathbb{R}$) if the input term $v(t)$ is replaced in Equation (2.5) by a suitable memoryless feedback function $K(x_1(t),...,x_n(t),u(t))$, and if a suitable memoryless readout function $h(\mathbf{x}(t))$ is applied to its internal state $\mathbf{x}(t) =< x_1(t),...,x_n(t) >$: one can achieve then that $h(\mathbf{x}(t)) = z(t)$ for any solution $z(t)$ of Equation (2.6).

Also the dynamic responses of all systems consisting of several higher order differential equations of the form Equation (2.6) can be simulated by fixed systems of the form (2.5) with a corresponding number of feedbacks.

The main result stated by Theorem 2.1 implies that an LSM defined by equations (2.8)-(2.9) can theoretically approximate any dynamical system arbitrarily close by only optimizing the readout parameters. In a follow-up paper [14], it is argued that introducing a state-feedback that improves the dynamic response of the arbitrary reservoir offers a much stronger universal approximation property compared to standard feedforward networks, which seems to undermine the results in Theorem 1. From the perspective of Control Theory, the Universal Approximation property of RC stated in Theorem 1 cannot hold only by training the readout. Take for example, a linear input-state-output system. Changing only the output matrix does not modify the dynamics of the system, which inherently, sets some boundaries to how close can such a system approximate the behaviour of another system with different dynamics and similar structure.

Theorem 2.2 that is presented in the form of Theorem 1 in [14] states that a dynamical system of the form given in (2.6) can perform highly accurate tasks and outperforms the standard LSM architecture provided by equations (2.8)-(2.9), as a state-feedback gain is able to modify the dynamics of the initial reservoir to be better suited for a particular target task. However, the major drawback of the proposed randomly initialized state-feedback LSM is that the state-feedback gain can only be trained when a priori information is known about the target system or via an estimation method.

Similar to Theorem 2.1, a number of approximation theorems and properties have been proposed for generic ANNs:

1. Kolmogorovs superposition theorem [71] is one of the first results in universal approximation theory and states that every continuous function can be represented as a superposition of continuous functions of 3 variables.

2. Cybenko in [72] proposed approximating arbitrary functions using a finite sum of sigmoidals.

3. Hornik in [73] showed that multilayer FNNs are universal approximators.

4. Leshno in [74] and Pinkus in [75] showed that the universal approximation property is equivalent to having a nonpolynomial activation function for the neurons. It was shown that

there exists a sigmoidal function such that two hidden layer NNs with bounded number of neurons are universal approximators.

5. For arbitrary depth networks, a number of authors [76],[77],[78],[79] contributed to the result obtained by Park [80] where the minimum network width required for universally approximating L^p functions using FNNs has been determined.

6. The authors in [81] proved that single hidden layer networks with bounded width are still universal approximators for univariate functions. This property does not stand when it comes to approximating multivariable functions.

2.3 Liquid State Machines

The Liquid State Machine is a type of Reservoir Computer proposed by Maass [58] which implements a recurrent Spiking Neural Network (SNN) reservoir. The main motivation behind LSMs was to develop a biologically plausible model of computation to describe cortical circuits [8].

2.3.1 LSM Architectures

A typical LSM consists of a fixed SNN that is connected to a bank of memory-less filters and an adjustable readout. The spiking neurons that form the reservoir are randomly initialized and do not require training. The trainable readout is linear in the parameters, represented as a weighted sum between the states of the reservoir.

The spike trains are elements of the following space [8]:

$$\mathbb{S}_0 = \{s | s = \{t_k\}_{k=1}^{P}, t_{k+1} > t_k \geq 0, \forall k = 1, ..., P-1\} \tag{2.7}$$

where $t_1, ..., t_P$ are distinct times, P the total number of spike occurrences.

Consider the following liquid [8]:

$$x^M(t) = F(L^M(u(t))) \tag{2.8}$$

with the readouts equation:

$$y(t) = f^M(x^M(t)) \tag{2.9}$$

where $u(\cdot) : \mathbb{R} \to \mathbb{R}^{N_{in}}$ is the input injected in the network, $L^M : \mathbb{S}_0^{N_{in}} \to \mathbb{S}^N$ is the liquid, $x^M(\cdot)$ are the states of the liquid, $y(\cdot)$ is the output of the liquid and N_{in} and N are the number of inputs and number of neurons in the SNN, respectively.

Definition 2.5 *The bank of memory-less filters $F : \mathbb{S}_0^N \to L^2(\mathbb{R})^N$ are included in a class of CB filters (e.g., delay filters, linear filters with impulse responses of the form $h(t) = e^{-at}, a > 0$ defined as in [15]) that have the point-wise separation property, as per [15],[31] if $\forall u(\cdot), v(\cdot) \in U^n, u(s) \neq v(s)$ for some $s \leqslant 0$, $\exists B \in CB$ such that $(Bu)(0) \neq (Bv)(0)$.*

In other words, from Definition 2.2, the equation of the memory-less delay filters F on a spike

train $s_n = \{t_k^n\}_{k=1}^P$, as defined in [32]:

$$F(s_n) = \sum_{k=1}^{P_n} e^{-\frac{t-t_k^n}{\tau_s}} \cdot 1_{[t_k^n,\infty)}(t) \tag{2.10}$$

where P_n denotes the number of spikes in s_n, $1_{[t_k^n,\infty)}$ denotes the characteristic function on the interval $[t_k^n,\infty)$ and $\tau_s \in \mathbb{R}_+$ is the time constant that describes the characteristics of the filtering circuit (i.e. it interprets the filters response in the time domain, and relates to the time it takes for the response to reach a defined percentage of the final value).

Definition 2.6 *The readout function $f^M(\cdot) : L^2(\mathbb{R})^N \to L^2(\mathbb{R})$ is chosen to be from an arbitrary class CF of functions, from [15], having the approximation property: $\forall m \in \mathbb{N}$, any compact set $X \subseteq \mathbb{R}$, any continuous function $h : X \to \mathbb{R}$, and $\forall \rho > 0$, $\exists f \in CF : |h(x) - f(x)| \leqslant \rho, \forall x \in X$.*

The most common readout is the linear unit:

$$y(t) = f^M(x(t)) = \sum_{n=1}^{N} w_n x_n(t) \tag{2.11}$$

A more detailed analysis in [8] showed that the new proposed architecture solved the problems of traditional RNNs regarding training and scalability by only training the readout module.

In [16], Maass has analysed and compared different implementations of modeling networks of biological neurons such as Markov Chains [82] or Boltzmann machines [83] with the LSM architecture [8] on the classic travelling salesman problem. The results show that the superior computational power of the spiking architecture comes from the fact that the transition between the states of the reservoir are substantially fewer for finding the optimal solution compared to the other architectures. Moreover, the afferent stochastic training strategies of SNNs offer improved energy consumption and faster running times compared to the traditional computing machines.

The shortcoming of this first generation of LSM architectures is that, in practice, the state of the liquid is uniformly sampled and that $x^M(\cdot)$ is not bandlimited. This means that the samples of the states are affected by aliasing, due to Shannons law, which leads to imprecise predictions of the final output.

One new architecture has been proposed in [32] which solves the limitations regarding the fitting imprecision of the last generation architecture, by eliminating the bank of filters F, by using a spike time based readout and by using a more appropriate spike space.

The spike trains are mapped in the extended Carnell-Richardson spike train space [84]:

$$\mathbb{S} = \{s = \{(a_k, t_k)\}_{k=1}^P, P \geqslant 1, t_k, a_k \in \mathbb{R}, t_k \neq t_l, \forall k, l \in \{1, ..., P\}, k \neq l\} \tag{2.12}$$

with the following three operations:

- **Vector sum:** $s_1 + s_2 = \{(a_k^1, t_k^1)\}_{k=1}^{M_1} \cup \{(a_k^2, t_k^2)\}_{k=1}^{M_2}$
- **Scalar Multiplication:** $\alpha \cdot s = \{(\alpha \cdot a_k, t_k)\}_{k=1}^{M}, \forall \alpha \in \mathbb{R}$
- **Inner Product:** $\{s_1, s_2\}_{\mathbb{S}} = \sum \sum_{k_1=1, k_2=1}^{k_1=M_1, k_2=M_2} a_{k_1}^1 a_{k_2}^2 \cdot e^{-\frac{|t_{k_1} - t_{k_2}|}{\tau_s}}$

It is preferred to use the Carnell-Richardson inner product space, compared to the original space of all possible spikes defined in (2.7) as the initial space proposed in [8] does not form a linear space because the condition for the scalar multiplication operation cannot be met:

$$\alpha \cdot s = \{(\alpha, t_k)\}_{k=1}^{M} \notin \mathbb{S}_0$$

In the newly proposed architecture, the bank of filters F is removed from the state equation in (2.8) whilst the output equation with the modified spiking readout structure $f^M(\cdot) : \mathbb{S}^N \to L^2(\mathbb{R})$ is in the following form:

$$y(t) = f^M(x^M(t)) = R_w s^{out} = \sum_{n=1}^{P_n^{out}} w_n s_n^{out} = s_w^y \tag{2.13}$$

where $s^{out} = [s_1^{out}, ..., s_n^{out}]$ and P_n^{out} denotes the number of spikes.

2.3.2 Spiking Neuron Models

The spiking neuron model has been introduced as a biological plausible mathematical model that describes, to some extent, the functionality of a real neuron. The operating principle of all spiking models is that the membrane voltage of the neuron evolves under the effect of the input. When a certain threshold is met, a spike is released and the neurons membrane voltage is reset to a default value. Compared to traditional non-spiking neurons, temporal information is encoded within the spikes that are released by triggering the spiking neurons. This grants SNNs more computational power at a fraction of the energy cost used by ANNs [85].

2.3.2.1 Integrate and Fire model

The Integrate-and-Fire neuron [86] is the simplest, most commonly used mathematical model of a biological neuron [8],[24],[32],[87], given by:

$$C_m \frac{dV_m(t)}{dt} = I(t) \tag{2.14}$$

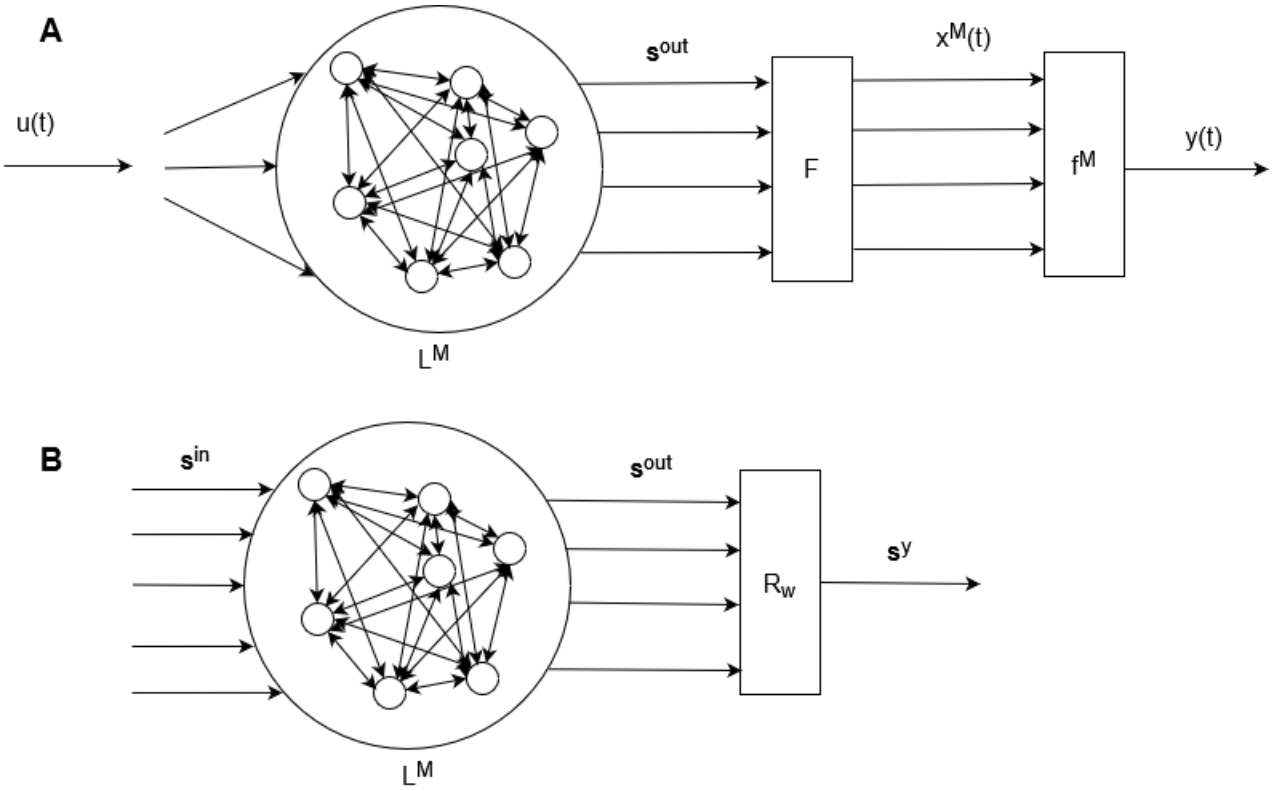

Figure 2.2: **LSM architecture generations**: A) First generation continuous-time output response LSM architecture B) Second generation spiking output response LSM architecture

where C_m is the membrane capacity and V_m is the membrane voltage.

The condition of firing is given by:

$$\frac{1}{C_m}\int_{t_k}^{t_{k+1}} I(t)dt = \Delta \tag{2.15}$$

where $\{t_k\}$ is the spike time sequence.

The output function is of the following form:

$$y(t) = \sum_k \delta(t - t_k) \tag{2.16}$$

where $\delta(t)$ is the Dirac delta function and $\Delta \in \mathbb{R}_+$ is the membrane voltage threshold.

The shortcoming of this neuronal model is represented by the lack of a time-dependent memory, meaning that the membrane voltage does not decay with time [88]. Past and current information have the same effect on the neurons output, which is unrealistic from a biological point of view. Naturally, past information should become less relevant, compared to current input signals that should have the highest impact on the neurons response.

2.3.2.2 Leaky Integrate-and-Fire model

The Leaky Integrate-and-Fire (LIF) neuron [24],[32],[89] has been introduced as an improvement to the IAF model by adding a time-dependent memory in the form of a leaky resistor, which overcomes the main drawback that the previous generation of neurons had in modelling the membrane voltage.

The improved equation is presented as following:

$$C_m \frac{dV_m(t)}{dt} = I(t) - \frac{V_m(t)}{R_m} \tag{2.17}$$

where R_m is the membrane resistance.

The condition of firing is given by:

$$\frac{1}{C_m} \int_{t_k}^{t_{k+1}} I(t) - \frac{V_m(t)}{R_m} dt = \Delta \tag{2.18}$$

where $\{t_k\}$ is the spike time sequence and $\Delta \in \mathbb{R}_+$ is the membrane voltage threshold.

The output function is of the following form:

$$y(t) = \sum_k \delta(t - t_k) \tag{2.19}$$

where $\delta(t)$ is the Dirac delta function.

2.3.2.3 Resonate and Fire model

This model is obtained by linearising the non-linear neuron membrane potential function [90]:

$$\begin{aligned} \dot{x} &= bx - \omega y \\ \dot{y} &= \omega x + by \end{aligned} \tag{2.20}$$

which has the following equivalent complex form:

$$\dot{z} = (b + i\omega)z \tag{2.21}$$

where $z = x + iy \in \mathbb{C}$ describes the oscillatory activity of the neuron, x is the current variable, y is the voltage variable, $\omega, b \in \mathbb{R}$.

2.3.2.4 Hodgkin-Huxley model

The most important and biological relevant neuron model has been proposed by Hodgkin and Huxley [91] in 1952. The highly complex model [92] focuses on describing the relationship between the ion currents crossing the neural membrane and the membrane voltage, with several applications in pattern recognition using LSMs [93],[94],[95],[96].

The models set of equations:

$$\begin{aligned} C_m\frac{dV_m}{dt} &= I-\bar{g}_K n^4(V_m-V_K)-\bar{g}_{Na}m^3h(V_m-V_{Na})-\bar{g}_l(V_m-V_l)\\ \frac{dn}{dt} &= \alpha_n(V_m)(1-n)-\beta_n(V_m)n\\ \frac{dm}{dt} &= \alpha_m(V_m)(1-m)-\beta_m(V_m)m\\ \frac{dh}{dt} &= \alpha_h(V_m)(1-h)-\beta_h(V_m)h \end{aligned} \tag{2.22}$$

with

$$\begin{aligned} &\alpha_n(V_m)=\frac{0.01(10-V_m)}{e^{\frac{10-V_m}{10}}-1};\alpha_m(V_m)=\frac{0.1(25-V_m)}{e^{\frac{25-V_m}{10}}-1};\alpha_h(V_m)=0.07e^{-\frac{V_m}{20}}\\ &\beta_n(V_m)=0.125e^{-\frac{V_m}{80}};\beta_m(V_m)=4e^{-\frac{V_m}{18}};\beta_h(V_m)=\frac{1}{e^{\frac{30-V_m}{10}}+1} \end{aligned} \tag{2.23}$$

where α_i,β_i are rate constants for the i-th ion channel, which depend on voltage and not time, n,m,h are dimensionless quantities between 0 and 1 that are associated with the potassium and sodium channels, $\bar{g}$ is the maximal value of the conductance and I is the injected input.

Other simplified Hodgkin-Huxley (HH) models have been developed, such as the Morris-Lecar (ML) [97], Poisson-Nerns-Planck (PNP), Poisson-Boltzmann-Nernst-Planck (PBNP) [98] and FitzHugh-Nagumo (FHN) [99], which aim to reduce the complexity of the HH model by reducing the order of the non-linear differential equations.

2.3.2.5 Morris-Lecar model

The ML model [97] proposes a differential equation where the inward current is calcium, rather that the sodium current modelled by the HH. Bursting was implemented by including an

intracellular pool of calcium and a second calcium-dependent potassium conductance:

$$\begin{aligned} C_m \frac{dV_m}{dt} &= I - g_{Ca} m_\infty (V_m - E_{Ca}) - g_K n (V_m - E_K) - g_L (V_m - E_L) - g_{K(Ca)} \frac{[Ca]}{[Ca]+K} (V_m - E_K) \\ \frac{d[n]}{dt} &= \frac{n_\infty - n}{\tau_n} \\ \frac{d[Ca]}{dt} &= k_1 I_{Ca} - k_2 [Ca] \end{aligned} \tag{2.24}$$

where $I_{K(Ca)}$ is the calcium-dependent current, $[Ca]$ is the calcium concentration.

2.3.2.6 FitzHugh-Naguomo model

The FHN model [99] is a simplification of the HH model by describing only two variables $v(t)$, which is the voltage, and $w(t)$, which is a recovery variable:

$$\begin{aligned} \varepsilon \frac{dv}{dt} &= F(v) - w + I \\ \frac{dw}{dt} &= v - \gamma w \\ F(v) &= v(1-v)(v+a) \end{aligned} \tag{2.25}$$

where $\varepsilon << 1, a, \gamma$ are constants and I is the injected current.

2.3.2.7 Hindmarsh-Rose model

This model [90] is based on the FHN model:

$$\begin{aligned} \frac{dx}{dt} &= y - f(x) - z + I \\ \frac{dy}{dt} &= g(x) - y \\ \frac{dz}{dt} &= \varepsilon (h(x) - z) \\ f(x) &= ax^3 - bx^2 \\ g(x) &= c - dx^2 \end{aligned} \tag{2.26}$$

where ε is a time scale of the slow adaptation current and $h(x)$ is the scale of influence of the slow dynamics that determines how the neuron fires.

2.3.2.8 Izhikevich model

Finally, the model proposed by Izhikevich [100] is given by:

$$\begin{aligned} &\tfrac{dv}{dt} = 0.04v^2 + 5v + 140 - u + I \\ &\tfrac{du}{dt} = a(bv - u) \\ &v \geq 30mV, v \leftarrow c, u \leftarrow u + d \end{aligned} \tag{2.27}$$

where a, b, c, d are parameters, v is the membrane potential of the neuron and u is the recovery variable of the membrane.

An in-depth analysis of the aforementioned models has been made in [96] for a facial recognition task. The results revealed that the ML model overall performs best in recognizing 8 different emotions (surprise, happiness, disgust, fear, anger, sadness, neutral). For disgust, it has been observed that the IF model is superior to the other models, whilst the FHN performed best for the classification of the neutrality expression. Although the Izhikevich model performed weakest overall, it had the best accuracy for classifying sadness. From these results, the authors concluded that there is no model that is clearly superior and that each particular task can yield for a particular neuron model to obtain maximum performance.

Lazar [101] proved that the IAF neurons are input-output equivalent with the HH models. By introducing a variable threshold sequence, that is identical to the inter-spike interval sequence generated by the HH, it allows for the more simple IAF model to generate spikes at the exact same spiking times as the HH. This means that the multiplicative coupling of the IAF is I/O equivalent: for the same input, the same output is generated. This represents a major breakthrough in modelling biological neurons as IAFs offer better numerical tractability and scalability by reducing the complexity of HH architectures whilst maintaining the same output behaviour.

2.3.3 LSM Training Algorithms

2.3.3.1 Standard approach

As LSMs fall under the RC framework hypotheses and do not require full training, there is a direct solution for finding the optimal readout, in the sense of LS [59],[102].

The first step is to randomly initialize the reservoir L^M and ensure that it possesses input separability and fading memory, based on a set of hyper parameters such as connection topology and reservoir dimensionality. As mentioned in earlier sections, genetic algorithms are useful in

generating these kind of successful topologies.

Consider a non-linear dynamical target system:

$$\begin{aligned} \dot{x} &= f(x,u) \\ y_{target} &= g(x) \end{aligned} \tag{2.28}$$

where $x : \mathbb{R} \to \mathbb{R}^N$ the states of the target, $f : \mathbb{R}^N \times \mathbb{R}^{N_{in}} \to \mathbb{R}^N$ a non-linear dynamical function, $y_{target} : \mathbb{R} \to \mathbb{R}^{N_{out}}$ the target output, $g : \mathbb{R}^N \to \mathbb{R}^{N_{out}}$ the non-linear output function.

Next, from the input sequence $u : \mathbb{R} \to \mathbb{R}^{N_{in}}, u(0), ..., u(n_{max})$, the network is stimulated to generate states for post-processing, in order to train the readout to match the desired output $y_{target}(0), ..., y_{target}(n_{max})$ of system (2.28), with $n_{max}+1$ number of training data points.

For the most common readout layer, i.e., the linear readout $f^M(x^M(t)) = W_{out}x^M(t)$, the solution is computed directly:

$$W_{out} = (S^{-1}D)^T \tag{2.29}$$

where $S = [x^M(0), ..., x^M(n_{max})]$ is a matrix in which each row is composed of the state vector for all the n_{max} training points and $D^T = \begin{bmatrix} y_{target}(0) \\ ... \\ y_{target}(n_{max}) \end{bmatrix}$.

2.3.3.2 Parameter selection approach

One new approach for training LSMs is presented in [32] in the form of an Orthogonal Forward Regression for Spike Trains (OFRST) algorithm. The proposed method relies on precise spike timing to select the neurons that are relevant to each learning task.

The problem addressed is to learn a target output function $y^*(t)$, given a N-dimensional liquid that generates outputs $\{s_k^{out}\}_{k=1}^N$ in response to input stimuli $\{s_k^{in}\}_{k=1}^{N_{in}}$. The algorithm proceeds, in a greedy fashion, by selecting the spike trains that are most relevant for the learning task using an OFR algorithm [23] applied on spike trains.

Start from the complete set of SNN outputs $s_k^{\perp(1)} = s_k^{out} \in \mathbb{S}, \forall k \in \{1,2,...,N\}$ and select the most significant spike train $s_{l_1}^{out}$ that maximizes the error-reduction-ratio, which expresses the effect of each spike fired by the neurons of the reservoir with respect to the target output:

$$ERR_k^{(i)} = \frac{\langle s_k^{\perp(i)}, s^{y*} \rangle_{\mathbb{S}}^2}{||s_k^{\perp(i)}||_{\mathbb{S}}^2 ||s^{y*}||_{\mathbb{S}}^2} \tag{2.30}$$

where s^{y*} is the target spike train and $s_k^{\perp(i)}$ is the complete set of outputs of the LSM, orthogonal on

the most significant spike train $s_{l_i}^{out}$ at step i.

The algorithm works iteratively by orthogonalising the remaining spike trains using a Gram-Schmitt routine and selects the next most significant spike trains. The following step is to generate the SNN outputs and corresponding output vector weights and evaluate the approximation performance of the current selection of parameters. The final step is to select the smallest number of neurons that lead to the maximum prediction accuracy on the validation dataset.

Experiments carried out by Florescu et. al. [32] show that for binary classification tasks, their proposed algorithm has the highest accuracy, compared to the state-of-the-art LS [59],[102] and OFR [23] algorithms while using the lowest number of readout connections.

2.3.3.3 The STDP application on LSM readouts

The introduction of the Spike-Timing-Dependent-Plasticity (STDP) rule [103],[104],[105] specifies how the synaptic efficacies should be adjusted, depending on the relative timing of presynaptic and postsynaptic neuron activations.

Paper [106] proposes a readout architecture in which, one readout circuit is connected to all the neurons in the liquid for each signal. Each connection from the neuron to the readout circuit has m different delay-weight paths.

The weight change of a synapse from a presynaptic neuron j to the postsynaptic neuron i:

$$\Delta w_j = \sum_{k=1}^{N} \sum_{n=1}^{N} g^M(s(t_i^n) - s(t_j^k)) \tag{2.31}$$

where $t_j^k, k = 1,2,3,...$ are the presynaptic spike times, $t_i^n, n = 1,2,3,...$ are the postsynaptic times.

The function $g^M(x^M)$ is defined as:

$$g^M(x^M) = \begin{cases} A_+ e^{-\frac{\Delta t}{\tau^2}}, \Delta t < 0 \\ -A_- e^{\frac{\Delta t}{\tau^2}}, \Delta t > 0 \end{cases} \tag{2.32}$$

where $A_+, A_- \in \mathbb{R}$ and $\tau = 2ms$ controls the time amplitude of the modification window.

2.4 Echo State Networks

Echo State Networks (ESNs) are non-spiking RNNs [9],[12],[107],[108] that emerged at the same time as LSMs [15]. It is an approach to RNNs based on generating a large randomly and sparsely connected neural network where only a single layer of output weights from the reservoir are trained as the target function. Different to LSMs, ESNs use non-spiking activation functions (e.g. linear functions, sigmoidal functions, etc.)

2.4.1 ESN Architectures

The basic discrete-time equations [18] of an ESN are given by:

$$x(k+1) = f(Wx(k) + W_{in}u(k) + W_{fb}y(k)) \tag{2.33}$$

and the output equation follows as:

$$y(k) = H(x(k), W_{out}) \tag{2.34}$$

where $x : \mathbb{R}^n \to \mathbb{R}^n$ is the n-dimensional state of the system, $u : \mathbb{R} \to \mathbb{R}^l$ is the l-dimensional input of the system, $y : \mathbb{R}^n \to \mathbb{R}^p$ is the output of the system, $f : \mathbb{R}^n \to \mathbb{R}^n$ is the reservoir activation function $W \in \mathbb{R}^{n \times n}$ is the reservoirs connection matrix, $W_{in} \in \mathbb{R}^{n \times l}$ is the input-to-reservoir connection matrix, $W_{fb} \in \mathbb{R}^{n \times p}$ is the fixed, randomly initialized output-feedback weight matrix and $H : \mathbb{R}^n \to \mathbb{R}^p$ is the output function. Generally, the readout is a linear function $H(x) = W_{out}x$, where $W_{out} \in \mathbb{R}^{p \times n}$ is the output weight matrix. The readout could also be represented by a linear-in-the-parameters non-linear map $H(x) = W_{out}g(x)$ with a set of basis functions in x, $g : \mathbb{R}^n \to \mathbb{R}^p$.

The non-spiking artificial neuron receives multiple inputs that are passed through a non-linear continuous function and produces an output. One of the most popular non-linear activation functions is the hyperbolic tangent function:

$$f : \mathbb{R} \to (-1, 1), f(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{2.35}$$

The major advantage of artificial neurons is that they facilitate the use of fast computational gradient descent training methods to modify the weights and biases of the NN, compared to the more complex, biological spiking neuronal models. However, the major drawback of this class of models is that they are incapable of accurately representing more complex behaviours that biological

neuronal models exhibit. Namely, the activity evoked by real neurons is irregular as they do not transmit information at each input propagation cycle, but only when the electrical potential inside the soma (the trunk in where the nucleus of the neuron lies and where the proteins are made to be transported throughout the dendrites and axons) reaches a certain threshold, whilst continuous and discrete non-spiking models propagate data at fixed cycles.

ESN architectures possess the echo state property, where the activations of the RNN neurons are systematic variations of the driver signal with respect to the spectral radius of W:

$$\rho(W) < 1 \tag{2.36}$$

From a control perspective, equation (2.36) ensures that the randomly initialized reservoir is stable when f is a linear function.

Various extensions and refinements of the original ESN model have been proposed in order to reduce the limitations imposed by a fixed reservoir regarding dynamics and improve the approximation performance of the current architectures. This includes ESN models with reservoirs that have uniformly distributed poles and adaptive bias [109], that aim to adjust the spectral radius of the fixed weight reservoir by only training the bias:

$$x(k+1) = f(Wx(k) + W_{in}u(k) + W_{in}b) \tag{2.37}$$

where $b \in \mathbb{R}$ is the adaptive bias.

For the processing of noncircular complex signals, augmented complex ESNs with non-linear readout layers [110] improve the approximation results over standard architectures. The non-linear readout offers more computational power compared to linear ones, whilst the augmented complex reservoir is more suited in processing complex inputs when compared to the standard reservoirs with real eigenvalues.

The echo state Gaussian process [111] is another architecture in which the high-dimensional feature projections are explicitly computed as the reservoir states, defined in (2.33).

The network output $\mathbf{y}(t)$ is considered to be consisting of M component responses $\mathbf{y}(t) = [y_j(t)]_{j=1}^{M}$, where each component:

$$y_j(t) = \mathbf{w}_j^T \phi(t) \tag{2.38}$$

where $\phi(t) = [\mathbf{x}(t); \mathbf{u}(t)]$.

Note that, the readout weights $\mathbf{w}_j$ are under a Gaussian distribution $\mathbf{w}_j \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

From this, it turns out that the ESN readout components yield a Gaussian Process (GP) form:

$$[y_j(t)]_{j=1}^{M} \sim \mathcal{N}(\mathbf{0}, \mathbf{K}_r(\phi, \Phi)) \tag{2.39}$$

where $\Phi = [\phi(t_1), ..., \phi(t_T)]$ is the design matrix and $\mathbf{K}_r$ is given by:

$$\mathbf{K}_r = \begin{bmatrix} \phi(t_1)^T\phi(t_1) & ... & \phi(t_1)^T\phi(t_T) \\ ... & & ... \\ \phi(t_T)^T\phi(t_1) & ... & \phi(t_T)^T\phi(t_T) \end{bmatrix} \tag{2.40}$$

where $\phi(t_1)^T\phi(t_2)$ are functions of the reservoir state vectors.

One other type of reservoir kernel is introduced as a Gaussian radial basis function:

$$k_r(\phi(t_1), \phi(t_2)) = e^{-\frac{||\phi(t_1)-\phi(t_2)||^2}{2\lambda^2}} \tag{2.41}$$

where $\lambda \in \mathbb{R}$ is a trainable parameter.

Training this structure is done by employing a conjugate gradient descent method [112].

The main disadvantage of using this type of architecture is that the approximations of the model covariance functions $\phi(t_i)^T\phi(t_j)$ is slow.

A more recent paper [113] introduces the concept of continuous-time ESNs. The state equation:

$$\dot{x}(t) = tanh(Wx(t) + W_{in}y(p^*, t)) \tag{2.42}$$

and the output equation:

$$y(t) = W_{out}x(t) \tag{2.43}$$

with the same definitions as in equations (2.33)-(2.34) and where P is a Cartesian space of parameters under which the model is expected to operate.

A new type of ESN equivalent architecture has been proposed in [19] called NG-RC. A feature vector is created directly from the sampled input data without the need of an NN. Namely,

$$\mathbb{O}_{total} = c \oplus \mathbb{O}_{lin} \oplus \mathbb{O}_{nonlin} \tag{2.44}$$

where $c \in \mathbb{R}$ is a constant, $\mathbb{O}_{lin}$ is the linear feature vector and $\mathbb{O}_{nonlin}$ is the non-linear part of the feature vector.

The linear feature vector at time i:

$$\mathbb{O}_{lin,i} = \mathbb{X}_i \oplus \mathbb{X}_{i-s} \oplus ... \oplus \mathbb{X}_{i-(k-1)s} \tag{2.45}$$

where $\mathbb{X}_i = [x_{1,i}, x_{2,i}...x_{d,i}]^T$ is a d-dimensional input vector and s is the space between observations.

The non-linear feature vector:

$$\mathbb{O}_{nonlin}^{(p)} = \mathbb{O}_{lin}\lceil\otimes\rceil\mathbb{O}_{lin}\lceil\otimes\rceil...\mathbb{O}_{lin} \tag{2.46}$$

where $\lceil\otimes\rceil$ is the operator that collects the unique monomials in a vector.

The output of the NG-RC:

$$\mathbb{Y}_{i+1} = \mathbb{W}_{out}\mathbb{O}_{total,i+1} \tag{2.47}$$

where $\mathbb{W}_{out}$ is the trainable output weight matrix that is trained with the conventional LS algorithm [59],[102].

The advantages of the new non-NN architecture include faster computational times, compared to standard RC architectures. This is due to smaller feature vector sizes, meaning that there are fewer trainable parameters that must be determined. Moreover, the size of the training data set could also be reduced as a consequence to the latter observation.

ESNs were primarily designed to model time series, but a recent study [114] showed that this network architecture can be also used for static data classification and clustering in the correlations between human emotions and brain activity.

2.4.2 ESN Training Algorithms

2.4.2.1 Echo State and FORCE learning

The presence of an output-feedback that feeds the output signal back into the ESNs reservoir shapes the network dynamics to better approximate the observed behaviour of the unknown dynamical target system through the trainable readout in the form of Echo-State [9] and First-Order, Reduced and Controlled Error (FORCE) learning [12],[108],[115]. The first training algorithm proposed by Jaeger [9] uses the target output as the feedback signal during training.

Thus, the state equation during training becomes:

$$x(k+1) = f(Wx(k) + W_{in}u(k) + W_{fb}y_{target}(k)) \tag{2.48}$$

The readout is computed via an LS, as in equation (2.29).

Later studies [12] showed that the Echo State training only worked when the initial errors between the target and output signals were small, otherwise, clamping the feedback with the target signal would usually lead to instability. The development of the FORCE algorithm [12] addressed this issue by removing the need to clamp the feedback and feeding the actual output of the network back into the reservoir. The goal of the algorithm is not to reduce the errors by a large amount, but

to keep the changes done to the feedback and readout as small as possible because large changes can drive the network into instability during the initial training phase.

We recall the notions from from [12] to underlie a more detailed overview of the FORCE algorithm:

The ESN equation:

$$\begin{aligned} \dot{\mathbf{r}}(t) &= h(W_{res}\mathbf{r}(t) + W_{in}u(t) + W_{fb}z(t)) \\ z(t) &= \mathbf{w}^T\mathbf{r}(t) \end{aligned} \tag{2.49}$$

where $u = [u_1, ..., u_m]^T, \mathbf{r} \in \mathbb{R}^n, z \in \mathbb{R}^p$ denote the input, state and output vectors, $h : \mathbb{R} \to \mathbb{R}^n$ is the reservoir activation function, $W_{res} \in \mathbb{R}^{n\times n}$ is the reservoir connection matrix, $W_{in} \in \mathbb{R}^{n\times m}$ is the input-to-reservoir connection matrix, $W_{fb} \in \mathbb{R}^{n\times p}$ is the output-feedback matrix, $\mathbf{w} : \mathbb{R}^{p\times n}$ are the output weights, the interval of time between modifications of the readout weights Δt and the desired target output $f(t)$.

The error prior to the weight update at time t is defined as:

$$e_-(t) = \mathbf{w}^T(t - \Delta t)\mathbf{r}(t) - f(t) \tag{2.50}$$

The main idea behind the training process of the FORCE methodology is to update the weights in such a way that the magnitude of $e_-(t)$ is reduced. The subsequent weight modification employs a Recursive Least Squares (RLS) algorithm [116],[6] that controls the magnitude of the error in a rapid and effective manner.

$$\mathbf{w}(t) = \mathbf{w}(t - \Delta t) - e_-(t)\mathbf{P}(t)\mathbf{r}(t) \tag{2.51}$$

where $\mathbf{P}(t) \in \mathbb{R}^{N\times N}$ is updated according to the following rule:

$$\mathbf{P}(t) = \mathbf{P}(t - \Delta t) - \frac{\mathbf{P}(t - \Delta t)\mathbf{r}(t)\mathbf{r}^T(t)\mathbf{P}(t - \Delta t)}{1 + \mathbf{r}^T(t)\mathbf{P}(t - \Delta T)\mathbf{r}(t)} \tag{2.52}$$

The initial value of P is taken to be

$$\mathbf{P}(0) = \frac{I_N}{\alpha} \tag{2.53}$$

where $\alpha \in \mathbb{R}$ is a constant parameter that acts as a learning rate. Small values of α imply fast learning of the FORCE algorithm, but, depending on the learning task, an α that is too small could make the weight changes so fast that the algorithm becomes unstable. Larger values of α imply slow learning. Values that are too large can lead the algorithm into not keeping the output close to

the target function, causing the learning to fail.

The new error after the update procedure:

$$e_+(t) = \mathbf{w}^T(t)\mathbf{r}(t) - f(t) \tag{2.54}$$

where $|e_+(t)| < |e_-(t)|$ with $\frac{|e_+(t)|}{|e_-(t)|} \to 1, t \to \infty$.

Using the readout to indirectly modify the dynamics of the reservoir is a compromise solution. Although the main goal of the output map is to closely mimic the output response of the target system, training the parameters will not lead to the optimal solution regarding output approximation for all types of tasks. This is due to the fact that through the output-feedback connection, the readout indirectly modifies the dynamics of the reservoir and these two aforementioned problems usually have distinct solutions. Moreover, it is not possible to train the output-feedback gain to compensate for the compromise made by the readout in finding the solution between output signal fitting and reservoir dynamical fitting as the two main problems are interconnected. The modification of the gain would lead to a modification of the readout for the output fitting task. The subsequent modification of the output map would lead to a modification of the gain in the reservoirs dynamical fitting problem. Theoretically, the modifications would oscillate indefinitely, without a palpable solution.

One improvement to current architectures provided by Sussillo [12], that can be observed in Figure 2.3 B, is to replace the output-feedback gain with a separate network that acts as a state-feedback gain. The only parameters suited for modification are in the form of the newly introduced networks input layer. Although this might be seen as an immediate improvement, in control theory, the dynamics of the secondary network are not modified by the input matrix. They are dictated by the system matrix, which in this case, is represented by all the inter-neural connections of the smaller network that are randomly initialized. Hence, the overall dynamics of the closed loop system are heavily biased by the dynamics of the smaller network. Moreover, without identifying the key dynamics of the target task, when there is no a priori knowledge, there is no practical solution of a precise initialization that would bridge the gap, in an effective manner, between the current dynamics of the closed-loop reservoir and the dynamics of the target. As a matter of fact, in the case of linear reservoirs, a random initialization could lead the closed-loop system to instability without placing appropriate constraints on setting the new eigenvalues of the resulting system.

The last solution presented in [12],[28], allow full training of the network, which will not be discussed in this thesis as it violates the main principle in RC - non-trainable reservoirs.

In the following Chapters, This thesis will present an extended mathematical analysis of the limitations of the current FORCE methodology and a solution that separates the two main problems via a trainable state-feedback gain ESN architecture.

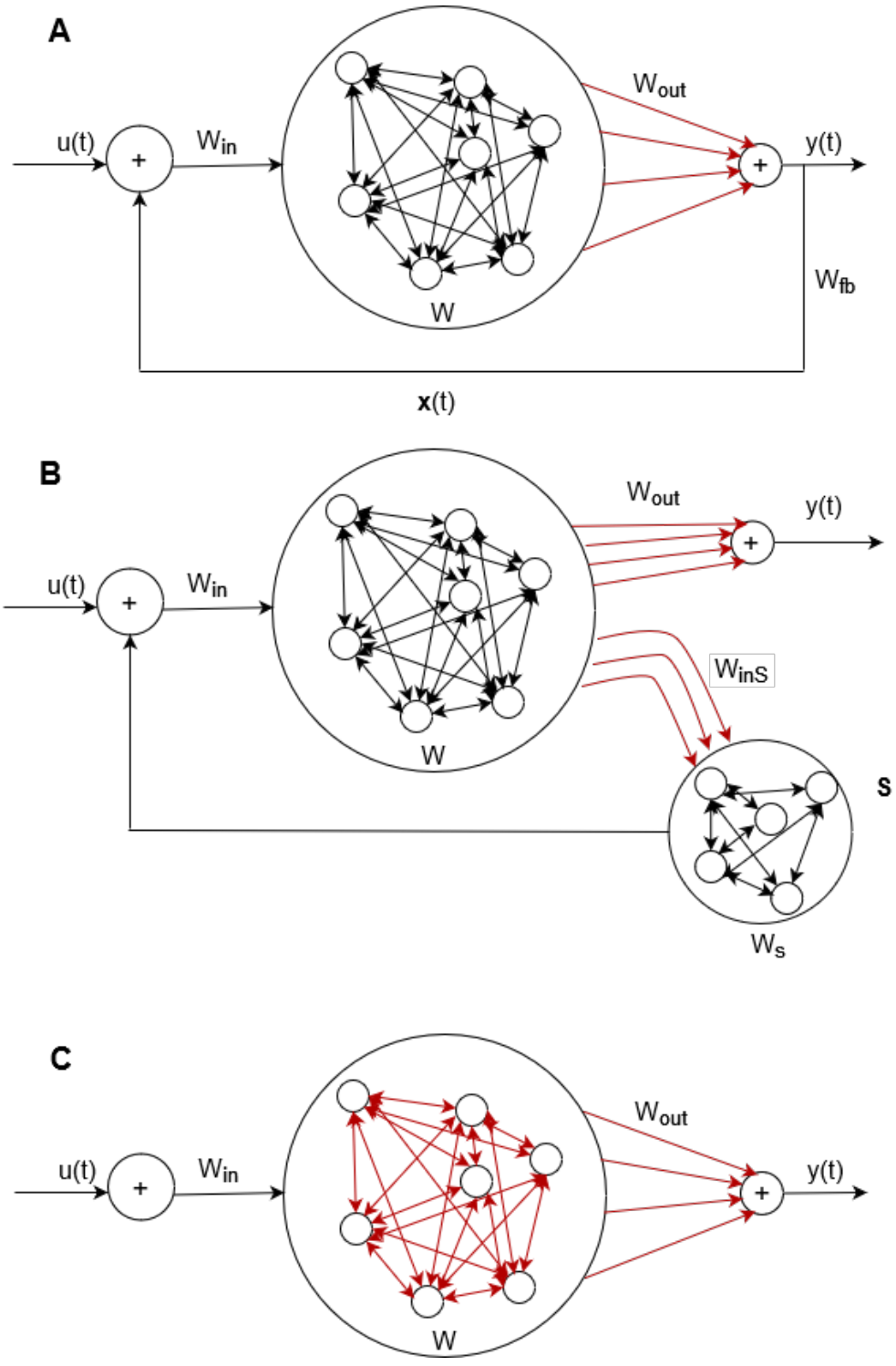

Figure 2.3: **FORCE ESN architectures. The red arrows represent the parameters of the network that are trained**: (A) Standard ESN architecture (B) State-feedback gain via a smaller NN ESN architecture (C) Fully trained network architecture

2.5 Conclusions

The major success of the RC framework, i.e., performing on par with fully trained RNNs by training only the readout of a randomly initialized feedback network architecture, reveals some important aspects:

- First of all, the feedback connection plays an important role in fitting the dynamics of the reservoir. The first improvement made in [14] by considering a state-feedback gain instead of an output-feedback gain should represent the way to go for current architectures. The state-feedback improves the approximation capabilities of RC architectures as the problems of output fitting and dynamical fitting would become separate. Explicitly, first training the state-feedback gain would allow the reservoir to represent the targets dynamics as close as possible. At a later stage, training the readout for fitting the target output should ensure maximum approximation of the overall structure. One downside to this approach can be represented by the fact that without a priori knowledge of the target task, it is difficult to find the optimum training strategy for the feedback weights. Thus, the existing training methods can be improved by using estimation methods such as the EKF [20],[21],[22] that would lead to optimum solutions for the state-feedback gain. This observation shall be further discussed in the following chapters.

- Secondly, introducing a trainable state-feedback gain into the current ESN architecture makes processing several tasks in parallel using the same reservoir impossible. This is due to the fact that each state-feedback gain is trained to minimize the differences in dynamics between each unknown target system and the closed-loop reservoir structure. As current developments in the literature have not considered a switching ESN structure for solving several tasks using state-feedback ESNs, it should be necessary and would be useful to analyse this new perspective for multiple-task computations.

- From the OFRST method proposed by [24] and its observations regarding low readout connectivity, it is clear that certain neurons are redundant when comes to output fitting. It should be interesting to see if an Orthogonal Forward Selection algorithm could be applied to the input layer and state-feedback gain in order to only select the most relevant nodes of the reservoir that should be modified to capture the important dynamics of a target task.

Based on the literature gaps identified in this chapter, the next chapter presents the theoretical limitations of classical Reservoir Computing implementations.

Chapter 3

Theoretical limitations of classical Reservoir Computing implementations

3.1 Introduction

This chapter revisits the original theoretical result concerning the computational power of classical reservoir computing implementations, i.e., the Universal Approximation property as stated in Theorem 1 [8] in Chapter 2, Section 2.2, under the assumption that only the parameters (weights) of the readout function are learned.

Specifically, it is shown that the original proof of the Universal Approximation Theorem of Reservoir Computing architectures in [8] does not hold.

This work is motivated by observing that in numerous numerical experiments the errors of approximation cannot be made arbitrarily small by learning only the output weights. To address these limitations is the reason why alternative RC architectures incorporating feedback loops and associated learning algorithms have been developed [9],[12],[14],[15],[16].

One important property possessed by the architectures within the Reservoir Computing framework is that of Universal Approximation, which states that any n-dimensional neural network

can approximate any n-dimensional system however close by only training the readout layer.

This chapter aims to conduct a more thorough theoretical analysis in which the conditions of applying the Stone-Weierstrass theorem of approximation underpinning the proof of the Universal Approximation property are analyzed. Our study shows that these conditions are not actually met, as training only the readout module does not guarantee operator separability, enforcing earlier observations in the literature that stated that current Reservoir Computing structures could be further improved.

In the following sections of this chapter, the reader will be presented with the short review of algebraic operators, theoretical limitations of the Universal Approximation property, some numerical studies and concluding remarks.

3.2 Notations and definitions

Let $\mathscr{U}$ be a compact metric space and

$$C(\mathscr{U},\mathbb{R}) \triangleq \{f : \mathscr{U} \to \mathbb{R}\} \tag{3.1}$$

with f continuous and equipped with the uniform norm:

$$||f||_\infty \triangleq \sup_{x\in[0,1]} |f(x)| \tag{3.2}$$

For any $f,g \in C(\mathscr{U},\mathbb{R})$ and $x \in \mathscr{U}$ we define fg by:

$$(fg)(x) = f(x)g(x) \tag{3.3}$$

It follows that $C(\mathscr{U},\mathbb{R})$ forms an algebra over $\mathbb{R}$.

Definition 3.1 *An algebra A is a vector space equipped with a bilinear product $m : A \times A \to A$. A is an unital algebra if $1 \in A$, where 1 is the identity element such that $1 \cdot f = f, \forall f \in A$.*

Definition 3.2 *A_Ψ is a subalgebra of A with generator $\Psi \subset A_\Psi$ if $\forall f,g \in A_\Psi, \alpha,\beta \in \mathbb{R}$ we have $\alpha f + \beta g \in A_\Psi$ and A_Ψ is the smallest subalgebra such that $\Psi \subset A_\Psi$.*

Definition 3.3 *A subalgebra $A_\Psi \subset C(\mathscr{U},\mathbb{R})$ separates points in $\mathscr{U}$ if for any $x,y \in \mathscr{U}, x \neq y$ there exists $f \in A_\Psi$ such that $f(x) \neq f(y)$.*

Definition 3.4 *Let F a field, and A be an F-vector space on which we define the vector product $(\cdot) : A \times A \to A$. Then A is called an algebra over F if $(A,+,\cdot)$ is a ring and $(\cdot)$ is bilinear.*

Definition 3.5 *A subalgebra A' is a subset of an algebra A, if A' is carrying the induced operations of A and is closed under all those operations.*

Stone-Weierstrass Theorem. *Let E be a compact metric space. If a subalgebra A of $\mathscr{C}_R(E)$ contains the constant functions and separates points on E, A is dense in the Banach space $\mathscr{C}_R(E)$.*

Definition 3.6 from [70]. *A non-linear operator $N : C(\mathbb{R}) \to C(\mathbb{R})$ has fading memory on a subset $\mathscr{U} \subset C(\mathbb{R})$ if there exists a decreasing function $\mu(t) : \mathbb{R}_+ \to (0,1], \lim_{t\to\infty}\mu(t) = 0$ such that for $\forall u \in \mathscr{U}$ and $\varepsilon > 0$ there exists $\delta > 0$ for $\forall v \in \mathscr{U}$ satisfying:*

$$\sup_{t\leq 0}|u(t)-v(t)|\mu(-t) < \delta \implies |(Nu)(0)-(Nv)(0)| < \varepsilon$$

In other words, the functional $\tilde{N}u(0) = (Nu)(0), \tilde{N} : \Pi\mathscr{U} \to \mathbb{R}$ is continuous with respect to the weighted norm, where $\Pi : C(\mathbb{R}) \to C(\mathbb{R}), \Pi u(t) = u(t), t \leq 0$. Moreover, μ will be called the weighting function and will say that N has w-fading memory.

Definition 3.7 *Let U_τ be the τ-delay operator defined by:*

$$U_\tau u(t) \triangleq u(t-\tau)$$

Definition 3.8 *A non-linear operator $N : \mathscr{U} \to C(\mathbb{R})$ is time-invariant if:*

$$U_\tau N u = N U_\tau u, \forall u \in \mathscr{U}, \tau \in \mathbb{R}$$

Definition 3.9 *N is causal if $u(\tau) = v(\tau)$ for $\tau < t$, which implies that:*

$$(Nu)(t) = (Nv)(t)$$

Here, $(Nu)(t)$ is a real valued functional $\tilde{N} : \mathscr{U} \to \mathbb{R}, \tilde{N}u(t) \triangleq (Nu)(t)$.

Definition 3.10 *N is continuous on $\mathscr{U}$ if for any $\varepsilon > 0$ there is a $\delta > 0$ such that for any $u, v \in \mathscr{U}$ the following condition is satisfied:*

$$||u - v|| < \delta \implies ||Nu - Nv|| < \varepsilon$$

Definition 3.11 Let $\mathscr{U} = \{u \in C(\mathbb{R}) \mid ||u||_\mu < M_1, ||U_\tau u - u||_\mu < M_2\tau, \tau \geq 0\}$ and $\mathscr{U}_- = \{\Pi u \mid u \in \mathscr{U}\}$ where $\mathscr{U}$ is bounded and $U_\tau : \mathscr{U} \to \mathscr{U}$ is bounded and continuous with respect to the $\mu-$weighted norm. For any $u \in \mathscr{U}, t \leq 0$, we have $u(t-\tau) = U_\tau u(t) \in \mathscr{U}$.

Definition 3.12 A subset A of a topological space X is said to be dense in X if every point of X either belongs to A or else is arbitrarily "close" to a member of A. Formally, A is dense in X if the smallest closed subset of X containing A is X itself.

3.2.1 Volterra Series Operator

Consider the non-linear dynamical system:

$$\begin{aligned} \dot{x} &= f(x,u) \\ y &= h(x) \end{aligned} \tag{3.4}$$

where $f : \mathbb{R}^n \to \mathbb{R}^n$ and $h : \mathbb{R}^n \to \mathbb{R}$ are smooth functions, $u \in C(\mathbb{R}_+)$ and $(x^*, u^*) \in \mathbb{R} \times \mathbb{R}$ is an equilibrium point for the system $f(x^*, u^*)$ such that the linearised system around the equilibrium point $\dot{x} = J_f(x^*, u^*)x$ is controllable and asymptotically stable and $x(t, t_0 = 0, x_0, u^*) \to x^*, \forall x_0 \in X \subset \mathbb{R}^n, x^* \subset X$, where J_f is the Jacobian, X is the domain of attractors of x^* and $x(t, t_0 = 0, x_0, u^*)$ is the solution of (3.4) at time t with initial condition $x(t_0) = x(0) = 0$ and input $u = u^* = ct$.

Let $\Psi = \{\psi_k\}_{k=1}^n, \psi_k : C(\mathbb{R}_+, \mathbb{R}) \to C(\mathbb{R}_+, \mathbb{R})$ with the system (3.4) with $y = y_k = x_k$.

It follows [117] that ψ_k has a Volterra Series representation:

$$(\psi_k u)(t) = w_0 + \sum_{i=1}^{\infty} \int_0^{\infty} \dots \int_0^{\infty} w_{i,k}(\tau_1, \dots, \tau_i) u(t-\tau_1) \dots u(t-\tau_i) d\tau_1 \dots d\tau_i \tag{3.5}$$

with $w_{i,k} \in L^1(\mathbb{R}_+^n)$ and

$$\int_0^{\infty} \dots \int_0^{\infty} w_{i,k}(\tau_1, \dots, \tau_i) u(t-\tau_1) \dots u(t-\tau_i) d\tau_1 \dots d\tau_i \leq \infty \tag{3.6}$$

3.3 The main theoretical results

Lemma 3.1 *Let $N_1, N_2 : \mathscr{U} \to C(\mathbb{R})$ be two non-linear operators with fading memory such that the associated functionals $\tilde{N}_1, \tilde{N}_2$ satisfy:*

$$|\tilde{N}_1 u(0) - \tilde{N}_2 u(0)| < \varepsilon, \forall u \in \mathscr{U}_-$$

It follows that:

$$||N_1 u - N_2 u||_\mu < \varepsilon, \forall u \in \mathscr{U}$$

Proof.

$$||N_1 u - N_2 u||_\mu = \sup_{t \leq 0} |N_1 u - N_2 u| \mu(-t)$$

Since $\mathscr{U}_-$ is compact [70], it follows by the extreme value theorem that $\exists t_1 \in (-\infty, 0]$:

$$\begin{aligned} \sup_{t \leq 0} |N_1 u - N_2 u| \mu(-t) &= |\tilde{N}_1 u(t_1) - \tilde{N}_2 u(t_1)| \mu(-t_1) = \\ &= |\tilde{N}_1 U_{-t_1} u(0) - \tilde{N}_2 U_{-t_1} u(0)| \mu(-t_1) \end{aligned}$$

Since $U_{-t_1} u(0) \in \mathscr{U}_-$ it follows that:

$$\begin{aligned} ||N_1 u - N_2 u||_\mu = |\tilde{N}_1 U_{-t_1} u(0) - \tilde{N}_2 U_{-t_1} u(0)| \mu(-t_1) \leq \\ \leq |\tilde{N}_1 U_{-t_1} u(0) - \tilde{N}_2 U_{-t_1} u(0)| < \varepsilon \end{aligned}$$

□

Let N be a non-linear, causal, time-invariant and continuous operator $N : C(\mathbb{R}) \to C(\mathbb{R})$ and $\tilde{N} : C(\mathbb{R}_-) \to \mathbb{R}$ such that the functional associated with N is defined by:

$$\tilde{N}\Pi u \doteq N u_e(0), u_e = \begin{cases} u(t) & t < 0 \\ u(0) & t \geq 0 \end{cases}$$

where $\Pi : C(\mathbb{R}) \to C(\mathbb{R}_-), \Pi u(t) = u(t), t \leq 0$

Theorem 3.1 *Let $\varepsilon > 0$ and $\{\psi_i\}_{i=1}, \psi_i : C(\mathbb{R}) \to C(\mathbb{R})$ be a set of operators with w-fading memory. Let $\{\tilde{\psi}_i\}_{i=1}, \tilde{\psi}_i : C(\mathbb{R}) \to C(\mathbb{R})$ be the set of functionals defined by:*

$$\tilde{\psi}_i u(t) = (\psi_i u)(t)$$

which are continuous with respect to the μ-weighted norm and separate points. Then there is a polynomial $\tilde{P}(\tilde{\psi}_1, ..., \tilde{\psi}_n) : \mathscr{U} \to \mathbb{R}$ and an associated polynomial operator $P : \mathscr{U} \to C(\mathbb{R})$ that:

$$a) |\tilde{N}u(t) - \tilde{P}(\tilde{\psi}_1, ..., \tilde{\psi}_n)u(t)| < \varepsilon$$

$$b) ||NU_t u - P(\psi_1, ..., \psi_n)U_t u||_\mu < \varepsilon$$

for all $u(t) \in \mathscr{U}$ and $t \in \mathbb{R}$.

Proof. Since the functionals $\tilde{\psi}_1, ..., \tilde{\psi}_n$ are continuous and separate points in $\mathscr{U}$ the subalgebra $\mathscr{A}_\Psi$ with generator $\Psi = \{\tilde{\psi}_1, ..., \tilde{\psi}_n\}$ over $\mathbb{R}$ is dense in $C(\mathscr{U}, \mathbb{R})$. It follows that for any $\varepsilon > 0$ there exist a polynomial $\tilde{P}(\tilde{\psi}_1, ..., \tilde{\psi}_n)$ such that:

$$|\tilde{N}u(t) - \tilde{P}(\tilde{\psi}_1, ..., \tilde{\psi}_n)u(t)| < \varepsilon$$

for all $u(t) \in \mathscr{U}$ and $t \in \mathbb{R}$.

Since $u(t) = U_t u(0) \in \mathscr{U}_-$ we have :

$$|\tilde{N}U_t u(0) - \tilde{P}(\tilde{\psi}_1, ..., \tilde{\psi}_n)U_t u(0)| < \varepsilon$$

From lemma 3.1 it follows that:

$$||NU_t u - P(\psi_1, ..., \psi_n)U_t u||_\mu < \varepsilon$$

for any $t \in \mathbb{R}$ and $u \in \mathscr{U}$.

Note that this does not imply that:

$$||Nu - P(\psi_1, ..., \psi_n)u|| < \varepsilon$$

where $||\cdot||$ is the uniform norm on $C(\mathbb{R})$, because $\mathscr{U}$ is not compact and so, we cannot apply the extreme value theorem.

□

Remark 3.1 The non-compactness of $\mathscr{U}$ in Theorem 3.1 is due to the fact that $\mathscr{U}$ is only defined on the μ-norm and not on the uniform norm [70].

3.4 LSMs limitation result

Theorem 3.2 *Consider the non-linear dynamical system:*

$$\begin{cases} \dot{x} = f(x,u) \\ y = x \end{cases} \tag{3.7}$$

where $f : \mathbb{R}^n \times \mathscr{U} \to \mathbb{R}^n$ is a smooth function, $u \in C(\mathbb{R})$ and (x^*,u^*) is an equilibrium point such that the linearised system around the equilibrium point $\dot{x} = J(x^*,u^*)x$ is controllable and asymptotically stable and let V_u be the domain of attraction of (x^*,u^*).

Let $\Psi = \{\psi_i\}_u^n$, $\psi_i : C(\mathbb{R}_+,\mathbb{R}) \to C(\mathbb{R}_+,\mathbb{R})$ be the input output maps associated to the system (3.7) with $y = x_i$ and $\{\tilde{\psi}_i\}$ the associated functionals $\tilde{\psi}_i : C(\mathbb{R}_+,\mathbb{R}) \to \mathbb{R}$ defined by:

$$\tilde{\psi}_i u(t) = (\psi_i u)(t)$$

which are continuous with respect to the μ-weighted norm.

It follows that the algebra generated by $\{\tilde{\psi}_i\}_{i=1}^n$ is not dense in $C(\mathbb{R})$.

Proof. Let $x_0 = x(t_0)$ and $x \in V$. Given that the system is controllable, then there exist $u_1, u_2 : (-\infty, T) \to \mathbb{R}$ such that:

$$u_1(t) = u_2(t), t \leq t_0, u_1 \neq u_2, t \in (t_0, T]$$

drive the system from initial state x_0 to x in finite time T along different state trajectories :

$$x_1(T,t_0,x_0) \neq x_2(T,t_0,x_0)$$

originating in x_0.

It is easy to see that since the system is controllable, for every $t \in (t_0,T)$, we can find $u_1, u_2 \in (-\infty, t)$ such that:

$$\gamma_1(t,t_0,x_0,u_1) = x_1(t,t_0,x_0)$$
$$\gamma_2(t,t_0,x_0,u_2) = x_2(t,t_0,x_0)$$

It follows that there are $u_1 \neq u_2 : (-\infty,T) \to \mathbb{R}$ such that $\tilde{\psi}_i(u_1(T)) = \tilde{\psi}_i(u_2(T)), i = 1,...,n$ for any $T > t_0$ and hence, the algebra generated by $\{\tilde{\psi}_i\}_{i=1}^n$ does not separate points in $C(\mathbb{R})$.

□

Remark 3.2 In other words, Theorem 3.2 shows that starting from an initial state, we can find two different control sequences that would lead the controllable system (3.7), to a final state, via two different state trajectories. Hence, the separability condition required to apply the Stone Weierstrass theorem of approximation is not met.

Theorem 3.3 *A liquid state machine defined by:*

$$\begin{cases} x(t) = (\Psi u)(t) \\ y(t) = h(x(t), W) \end{cases}$$

where $u(t) \in \mathscr{U} \subset C(\mathbb{R}), x \in \mathbb{R}^n, y \in \mathbb{R}, h : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ *is smooth and* $\Psi u = [\psi_1 u, ..., \psi_n u]$ *where* $\psi_i : C(\mathbb{R}) \to C(\mathbb{R})$ *are input-output operators with fading memory associated with the dynamical system:*

$$\begin{cases} \dot{x} = f(x, u) \\ y_i = x_i \end{cases}$$

where f *is a smooth functional,* $W = [w_1, ..., w_N] \in \mathbb{R}^n$ *is a vector of adjustable parameters, cannot approximate arbitrarily close any given non-linear, time invariant operator* $N : C(\mathbb{R}) \to C(\mathbb{R})$ *that has fading memory on* $\mathscr{U} \subset C(\mathbb{R})$.

Proof. Let's assume without loss of generality that the readout/output map is given by:

$$h(x(t), W) = \sum_{i=1}^{n} w_i g_i(x)$$

where $g : \mathbb{R}^n \to \mathbb{R}$ is a smooth function.

It follows that the subalgebra defined as $A_{\psi_g} \subset C(\mathscr{U}, \mathbb{R})$ with the generator function $G(u) = \{g_1(\Psi\tilde{(u)}), ..., g_n(\tilde{\Psi}(u))\}$ where $\tilde{\Psi} = [\tilde{\psi}_1, ..., \tilde{\psi}_n]$ and $\tilde{\psi}_i(u) = (\psi_i u)(t)$ are the functionals associated with the input output operators ψ_i does not separate points in $\mathscr{U}$ since we can find $u_1 \neq u_2 \in \mathscr{U}$ such that $\Psi(\tilde{u_1}(t)) = \Psi(\tilde{u_2}(t))$ and therefore

$$g_i(\Psi(\tilde{u_1}(t))) = g_i(\Psi(\tilde{u_2}(t))), i = 1, ..., n$$

This applies to every linear combinations of g_i and therefore $h(\Psi\tilde{(u_1)}, W) = h(\Psi\tilde{(u_2)}, W)$ for any choice of $W \in \mathbb{R}^n$.

□

Remark 3.3 Following Theorem 3.2 and Remark 3.2, Theorem 3.3 shows that LSMs cannot approximate arbitrarily close any given non-linear target function through a simple example. Because we can find two control sequences that lead us to the same final state via two different state trajectories, it is obvious that the readout function h will have the same results, even though the control sequences differ. Hence, it cannot separate points and thus, the Stone-Weierstrass theorem of approximation cannot be applied in proving that LSMs represent Universal Approximators.

3.5 Conclusions

This chapter has revisited a fundamental thoeretical result that underpinned the assertion that basic reservoir computing architecture, consisting of a non-linear, randomly initialised dynamical reservoir and a readout map with tunable weights has universal approximation properties, if it is assumed that only the readout weights are modified through training. This applies to reservoir computing architectures with general linear or non-linear readouts so for example, the Universal Approximation property will not hold even if polynomial readouts of arbitrary order are considered. While increasing the reservoir dimension and using a non-linear readouts improves accuracy, the resulting RC models cannot achieve arbitrarily small approximation error even in a noise-free case.

The alternative is to either train the reservoir weights or modify the reservoir dynamics through feedback as shown in Maass and Sontag [31]. This strategy has been adopted by a number of authors [9],[12],[14],[15],[16] and forms the basis for the well known FORCE learning algorithm.

Along this line, the following chapter introduces a novel RC architecture with trainable state-feedback gains and input layers, and associated learning algorithms.

Chapter 4

A novel state-feedback training algorithm for ESNs

4.1 Introduction

Following the conclusions of the previous chapter, it is necessary to develop new types of RC architectures and algorithms in order to ensure system stability and enhanced processing power.

Recurrent neural networks are artificial neural networks [39],[118],[119],[120] that incorporate feedback connections. These enable RNNs to learn and predict complex dynamical behaviors [26],[27],[28] and thus provide plausible models of brain architecture and computation [121]. Historically, compared to standard feedforward neural networks, RNNs have been considered to be difficult to train [122] which has prompted the development of novel network architectures and training algorithms to address challenges of learning long-range dependencies.

Echo-state networks are RNNs that exploit the properties of the RC architectures and its efficient training methods [9],[17],[107],[123] to achieve performance comparable to that of fully trained RNNs while incurring a fraction of the computational cost associated with training a RNN in full.

Specifically, ESNs consists of a dynamic reservoir – typically a RNN with sigmoid neurons having fixed, randomly assigned synaptic weights - that maps the inputs $u : \mathbb{R} \to \mathbb{R}$ to a high-dimensional state-space $\mathbf{x} : \mathbb{R} \to \mathbb{R}^n$ [14], and a feedforward readout, with adjustable synaptic weights that maps the reservoir state $\mathbf{x}$ to the network output $y : \mathbb{R} \to \mathbb{R}$. Since only the weights of the readout or output map are trained, an obvious advantage of RC is that training an ESN network is computationally efficient and fast compared to training all the parameters of the network [124]. Various extensions and refinements of the original ESN model have been proposed to improve performance including ESN models with state-feedback and reservoirs exhibiting scale-free and small-world characteristics [125], reservoirs with uniformly distributed poles and adaptive bias [109], augmented complex ESNs with non-linear readout layers [110], and echo state Gaussian process [111]. While usually the reservoir dynamics are non-linear and the readout is a linear map, an alternative architecture with linear dynamic reservoir and non-linear readout has also been considered [19],[67],[68],[69].

Typically, for a fully connected readout, the connection weights are trained using a LS optimization algorithm [8],[9],[12],[18],[108]. It has been shown that by selecting the readout connectivity as part of the optimization procedure, resulting to a sparse connectivity between the readout and reservoir neurons, improves accuracy [32]. This mirrors the existence in the brain of highly connected neurons with static connectivity, which implement generic functional roles in information processing, as well as of sparsely connected neurons exhibiting dynamic and plastic patterns of connectivity associated with task learning [126].

Alternative, more powerful methods for training RC network architectures involve shaping the dynamics of the reservoir through the introduction of a feedback loop , whilst still adjusting only the weights in the output layer. To avoid feeding back erroneous output to the reservoir in [9], the target output is fed back to the reservoir during training, instead of the actual network output. The network output is reconnected to the reservoir once the training phase has been completed. This approach works if the initial errors between the two output signals are small. If the initial errors are large, clamping the feedback usually leads to instability [108]. This issue is addressed by the FORCE learning approach [12] that eliminates the requirement to feed back the target output during training.

Whilst the introduction of a fixed output or state-feedback [14] shapes the dynamics of the system (i.e., the eigenvalues of the resulting closed-loop system) helping to bring the output closer to the target, optimizing the readout weights alone does not guarantee that these often competing goals can be achieved. In theory, matching the eigenvalue structure and the output map of a target dynamical system would require optimizing the feedback gains as well as the readout weights.

In this chapter, we propose a new method for training ESN with state-feedback which optimizes both the input layer - state-feedback gain pairs and readout weights, producing models

that significantly outperform ESNs in which only the readout weights are trained [8],[9],[12]. To that end we adopt the ESN architecture [109] consisting of a linear dynamic reservoir and a non-linear readout (polynomial) map. Training the input layer - state-feedback gain is realised using the EKF estimation method [20],[21],[22]. Training the readout implements the greedy optimization strategy proposed in [32] to identify the best synaptic connectivity for the readout module.

In the following sections of this chapter, the reader will be presented with the theoretical limitations of current output-feedback architectures, the advantages of full state-feedback ESNs, a novel training algorithm for state-feedback gain ESNs, some numerical studies and concluding remarks.

Specifically, the following section presents the theoretical limitations of the state-of-the-art output-feedback ESN training algorithms.

4.2 Theoretical limitations of output-feedback ESN training algorithms

A discrete-time ESN with m inputs, n units (neurons) and p outputs has the following state-space representation:

$$\begin{aligned} \mathbf{x}(k+1) &= f\left(W\mathbf{x}(k) + W_{in}\mathbf{u}(k) + W_{fb}y(k)\right) \\ y(k) &= H(\mathbf{x}(k), W_{out}) \end{aligned} \tag{4.1}$$

where $\mathbf{u} = [u_1, ..., u_m]^T, \mathbf{x}, \mathbf{y}$ denote the input, state and output vectors, $f : \mathbb{R} \to \mathbb{R}^n$ is the reservoir activation function, $W \in \mathbb{R}^{n\times n}$ is the reservoir connection matrix, $W_{in} \in \mathbb{R}^{n\times m}$ is the input-to-reservoir connection matrix, $W_{fb} \in \mathbb{R}^{n\times p}$ is the output-feedback matrix and $H : \mathbb{R}^n \to \mathbb{R}^p$ is the output (readout) function. Generally [8], $H(\mathbf{x}) = W_{out}\mathbf{x}$, where $W_{out} \in \mathbb{R}^{p\times n}$ is the output weight matrix. However, the readout could also be a linear-in-the-parameters non-linear map $H(\mathbf{x}) = W_{out}g(\mathbf{x})$, where $g : \mathbb{R}^n \to \mathbb{R}^p$ represents a set of basis functions in $\mathbf{x}$.

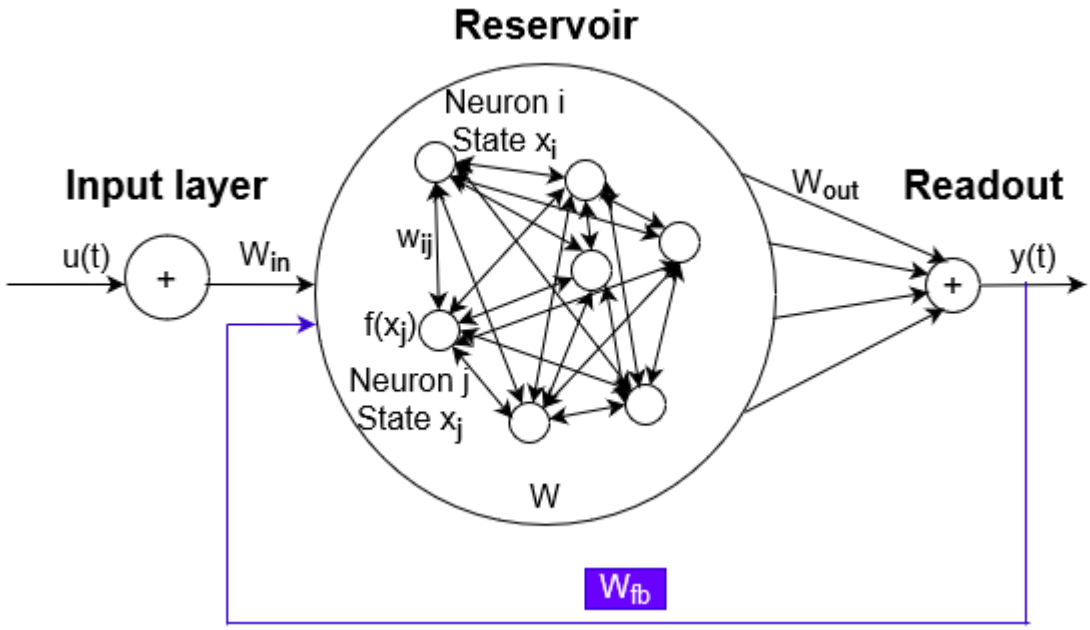

Figure 4.1: **Network architecture and adjustable weights** for FORCE training

In the following sections, both linear and non-linear activation functions will be considered, specifically the hyperbolic tangent function $f(x) = \frac{e^{2x}-1}{e^{2x}+1}$.

The FORCE algorithm provides a solution for training the readout weights W_{out} of an ESN with the fixed output-feedback loop presented in equation (4.1) using an RLS algorithm [12]. ESNs

with linear reservoirs and non-linear readout have also been considered. A linear-in-the-parameters non-linear readout map $H(\mathbf{x}) = W_{out}g(\mathbf{x})$ where g are multivariate monomials, for example, has the advantage that weights can still be trained using a least-squares algorithm.

Lemma 4.1 *Training only the output weights in equation (4.1) for a linear reservoir, does not guarantee that the ESN can approximate arbitrarily close a target dynamical system.*

Proof. Consider a SISO linear continuous-time ESN with output-feedback:

$$\begin{aligned} \dot{\mathbf{x}}(t) &= (W - W_{fb}W_{out})\mathbf{x}(t) + W_{in}u(t) \\ y(t) &= W_{out}\mathbf{x}(t) \end{aligned} \tag{4.2}$$

where $W \in \mathbb{R}^{n\times n}, W_{in} \in \mathbb{R}^{n\times 1}, W_{fb} \in \mathbb{R}^{1\times n}, W_{out} \in \mathbb{R}^{1\times n}$ are the reservoir connectivity matrix, input layer, state-feedback gain and linear readout.

The equivalent transfer function representation of equation (4.2) is given by:

$$T(s) = W_{out}(sI_n - (W - W_{fb}W_{out}))^{-1}W_{in} \tag{4.3}$$

Let us assume that the target system is described by the following state space model:

$$\begin{aligned} \dot{\mathbf{x}}_t(t) &= A\mathbf{x}_t(t) + Bu(t) \\ y(t) &= C\mathbf{x}_t(t) \end{aligned} \tag{4.4}$$

where $A \in \mathbb{R}^{n\times n}, B \in \mathbb{R}^{n\times 1}, C \in \mathbb{R}^{1\times n}$ are the system, input and output matrices.

The equivalent transfer function representation of target system is given by:

$$T^*(s) = C(sI_n - A)^{-1}B \tag{4.5}$$

Training the ESN to match the target is equivalent to finding W_{out}, W_{in}, W_{fb} that satisfy the functional equation $T(s) = T^*(s)$. If only the output weights W_{out} are trained, as in the case of FORCE, even if we assume $W_{in} = B$, in general we cannot find W_{out} to satisfy the following two equations:

The equation of dynamical fitting:

$$\det(sI_n - A) = \det(sI_n - (W - W_{fb}W_{out})) \tag{4.6}$$

The equation of output fitting:

$$W_{out}\mathbf{x} = C\mathbf{x} \tag{4.7}$$

where the unknowns are the output weights W_{out}.

In order to exactly match the target, matching the dynamics of the system is equivalent to solving (4.6) which has unique solution if the pair W, W_{in} is controllable. However, in general, the solution is $W_{out} \neq C$ and so the ESN output will not match the target. In practice, the reservoir dimension is larger than that of the target system and the FORCE algorithm achieves a good compromise in matching the dominant (slow) dynamics and the output of the target even when $W_{in} \neq B$. However, finding (training) the output as well as the feedback and input weights of the linear ESN would allow matching the dynamics of an arbitrary (stable) linear system of the same order.

More generally, an input affine non-linear ESN:

$$\begin{aligned} \dot{\mathbf{x}} &= f(\mathbf{x}(t)) + g(\mathbf{x}(t))u(t) \\ y &= h(\mathbf{x}(t)) \end{aligned} \tag{4.8}$$

can match the dynamics of an arbitrary non-linear system described by the input-output equation:

$$y^n(t) = G(y^{n-1}(t), y^{n-2}(t), ..., y(t)) + u(t) \tag{4.9}$$

by finding suitable state-feedback $u = K(\mathbf{x}(t), u(t))$ and readout $h(\mathbf{x}(t))$ functions [14].

□

To address some of the limitations presented above, the following section introduces full state-feedback ESNs with their advantages.

4.3 Full state-feedback ESN

Consider the alternative ESN architecture with state-feedback, illustrated in Figure 4.2:

$$\begin{aligned} \mathbf{x}(k+1) &= f\left((W - W_{in}W_{fb})\mathbf{x}(k) + W_{in}\mathbf{u}(k)\right) \\ y(k) &= H(\mathbf{x}(k), W_{out}) \end{aligned} \tag{4.10}$$

where $\mathbf{u} = [u_1, ..., u_m]^T, \mathbf{x}, \mathbf{y}$ denote the input, state and output vectors, $f : \mathbb{R}^n \to \mathbb{R}^n$ is the reservoir activation function, $W \in \mathbb{R}^{n\times n}$ is the reservoir connection matrix, $W_{in} \in \mathbb{R}^{n\times m}$ is the input-to-reservoir connection matrix, $W_{fb} \in \mathbb{R}^{n\times p}$ is the output-feedback matrix and $H : \mathbb{R}^n \to \mathbb{R}^p$ is the output (readout) function with $W_{out} \in \mathbb{R}^{p\times n}$ the output weight matrix.

Here we introduce an Alternate FORCE learning algorithm (aFORCE) for ESNs with trainable state-feedback gains. The algorithm is similar to the conventional FORCE [12] in the sense that it employs a similar strategy to control the magnitude of the error. The key difference in the case of aFORCE is that the output weights and the input layer - state-feedback gain pair are alternately trained. The learning process has two distinct components. First, an EKF algorithm [20],[21],[22] is employed to control the magnitude of the training error with respect to the state-feedback gain and input layer. Then the output (readout) weights are estimated with the feedback gains fixed. During the second phase of the algorithm, the number of input lags and the polynomial readout model structure are selected.

We have chosen the EKF estimation method for training the state-feedback gain and input layer as it conserves the advantage of the standard Kalmann filter which has been specifically designed for offering good estimation results for linear systems. The main drawback however, of the classical Kalmann filter is that it cannot be used for approximating non-linear systems. The EKF solves this issue by propagating the estimated state distribution through a first-order linearization of the non-linear system [127]. In general, one particular disadvantage of the EKF is that the goodness of the approximation is dependant on the degree of nonlinearity of the functions it approximates. Fortunately, in our case, using the hyperbolic tangent does not affect the precision of approximation as its Taylor series expansion is highly dependent on its first two terms ($x - \frac{x^3}{3} + \frac{2x^5}{15} + O(x^6)$).

Lemma 4.2 *aFORCE does not modify, in a direct manner, the weights of the reservoir and eliminates the severe limitation of the FORCE method of using the output layer for controlling the reservoirs dynamics.*

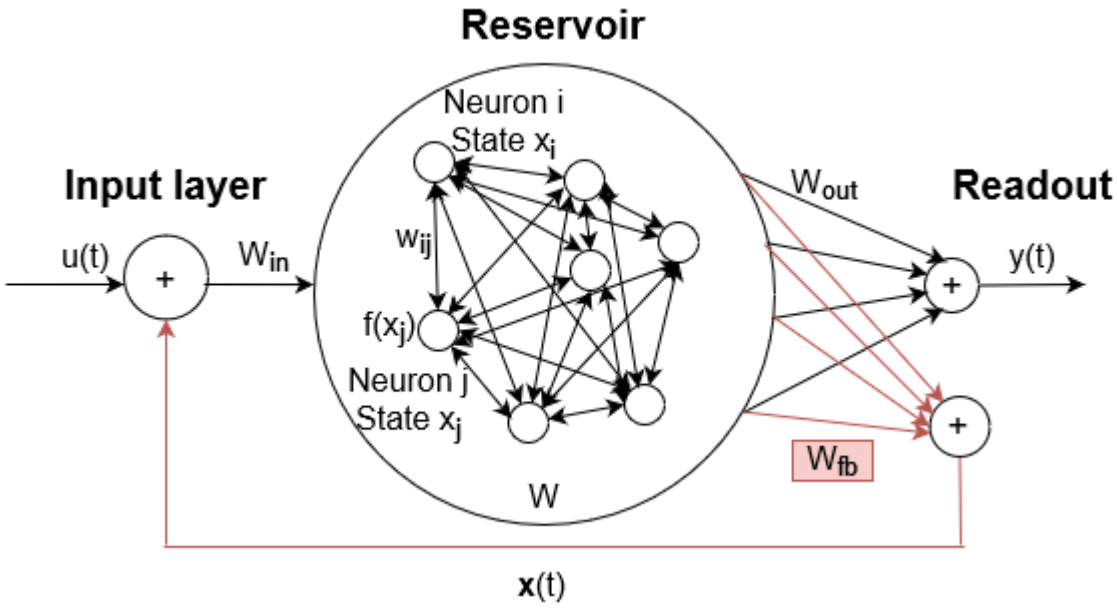

Figure 4.2: **Network architecture and adjustable weights** for aFORCE training

Proof. The state-space representation of the target system:

$$\begin{aligned} x_T(k+1) &= A_T x_T(k) + B_T u(k) \\ y_T(k) &= C_T x_T(k) \end{aligned} \tag{4.11}$$

where $n \in \mathbb{N}^*$ is the dimension of the system, $l \in \mathbb{N}^*$ the dimension of the input, $p \in \mathbb{N}^*$ the dimension of the output, $A_T \in \mathbb{R}^{nxn}$ the system matrix, $B_T \in \mathbb{R}^{nxl}$ the input matrix, $C_T \in \mathbb{R}^{pxn}$ the output matrix, $u : \mathbb{R} \to \mathbb{R}^l$ the input, $x_T : \mathbb{R} \to \mathbb{R}^n$ the states of the target and $y_T : \mathbb{R} \to \mathbb{R}^p$ the output of the target.

The state-space representation of the network:

$$\begin{aligned} x_N(k+1) &= (A_N + B_N K_N) x_N(k) + B_N u(k) \\ y_N(k) &= C_N x_N(k) \end{aligned} \tag{4.12}$$

where $A_N \in \mathbb{R}^{n \times n}$ is the reservoir weight matrix, $B_N \in \mathbb{R}^{n \times l}$ the input weight matrix, $K_N \in \mathbb{R}^{n \times p}$ the output-feedback matrix, $C_N \in \mathbb{R}^{p \times n}$ the output matrix, $x_N : \mathbb{R} \to \mathbb{R}^n$ the states of the network and $y_N : \mathbb{R} \to \mathbb{R}^p$ the output of the network.

Start from the Input-Output representation of systems (4.11) and (4.12).

The state and output representations of target (4.11):

$$X_T(s) = (A_T - sI_n)^{-1} B_T U(s) \tag{4.13}$$

$$Y_T(s) = C_T X_T(s) = C_T(A_T - sI_n)^{-1} B_T U(s) \tag{4.14}$$

The state and output representations of network (4.12):

$$X_N(s) = [(A_N + B_N K_N) - sI_n]^{-1} B_N U(s) \tag{4.15}$$

$$\begin{aligned} &Y_N(s) = C_N X_N(s), \\ &Y_N(s) = C_N[(A_N + B_N K_N) - sI_n]^{-1} B_N U(s) \end{aligned} \tag{4.16}$$

Target (4.14) can be dynamically approximated by the network (4.15) if and only if the Input-Output representations of the two systems coincide. Hence, the targets equations (4.13)-(4.14) would be equal to the networks equations (4.15)-(4.16).

Start with the output equality between (4.14) and (4.16):

$$\begin{aligned} &C_T X_T(s) = C_N X_N(s) \\ &C_T(A_T - sI_n)^{-1} B_T U(s) = C_N(A_N + B_N K_N - sI_n)^{-1} B_N U(s) | \frac{B_T^T}{||B_T||^2} \\ &C_T(A_T - sI_n)^{-1} = C_N(A_N + B_N K_N - sI_n)^{-1} \frac{B_N B_T^T}{||B_T||^2} | (A_T - sI_n) \\ &\frac{C_N^T}{||C_N||^2} | C_T = C_N(A_N + B_N K_N - sI_n)^{-1} \frac{B_N B_T^T}{||B_T||^2} (A_T - sI_n) \\ &(A_N + B_N K_N - sI_n) | \frac{C_N^T C_T}{||C_N||^2} = (A_N + B_N K_N - sI_n)^{-1} \frac{B_N B_T^T}{||B_T||^2} (A_T - sI_n) \\ &(A_N + B_N K_N - sI_n) \frac{C_N^T C_T}{||C_N||^2} = \frac{B_N B_T^T}{||B_T||^2} (A_T - sI_n) \\ &A_N \frac{C_N^T C_T}{||C_N||^2} + B_N K_N \frac{C_N^T C_T}{||C_N||^2} - sI_n \frac{C_N^T C_T}{||C_N||^2} = \frac{B_N B_T^T}{||B_T||^2} A_T - s \frac{B_N B_T^T}{||B_T||^2} I_n \\ &\begin{cases} A_N \frac{C_N^T C_T}{||C_N||^2} + B_N K_N \frac{C_N^T C_T}{||C_N||^2} = \frac{B_N B_T^T}{||B_T||^2} A_T \\ I_n \frac{C_N^T C_T}{||C_N||^2} = \frac{B_N B_T^T}{||B_T||^2} I_n \end{cases} \end{aligned} \tag{4.17}$$

For the presumption of a true dynamical approximation to hold, the following two equations must be satisfied at the same time:

$$A_N \frac{C_N^T C_T}{||C_N||^2} + B_N K_N \frac{C_N^T C_T}{||C_N||^2} = \frac{B_N B_T^T}{||B_T||^2} A_T \tag{4.18}$$

$$I_n \frac{C_N^T C_T}{||C_N||^2} = \frac{B_N B_T^T}{||B_T||^2} I_n \tag{4.19}$$

Equation (4.19) yields a direct solution:

$$\begin{aligned} C_T &= C_N \frac{B_N B_T^T}{||B_T||^2} I_n \\ C_N &= C_T B_T \frac{B_N^T}{||B_N||^2} \end{aligned} \tag{4.20}$$

Solution (4.20) yields a direct solution of equation (4.18):

$$
\begin{aligned}
&A_N \frac{\frac{B_N}{||B_N||^2} B_T^T C_T^T C_T}{||C_T B_T \frac{B_N^T}{||B_N||^2}||^2} + B_N K_N \frac{\frac{B_N}{||B_N||^2} B_T^T C_T^T C_T}{||C_T B_T \frac{B_N^T}{||B_N||^2}||^2} = \frac{B_N B_T^T}{||B_T||^2} A_T \\
&A_N \frac{B_N B_T^T}{||B_T||^2} + B_N K_N \frac{B_N B_T^T}{||B_T||^2} = \frac{B_N B_T^T}{||B_T||^2} A_T \\
&\frac{B_N^T}{||B_N||^2} | B_N K_N \frac{B_N B_T^T}{||B_T||^2} = \frac{B_N B_T^T}{||B_T||^2} A_T - A_N \frac{B_N B_T^T}{||B_T||^2} \\
&K_N \frac{B_N B_T^T}{||B_T||^2} = \frac{B_T^T}{||B_T||^2} A_T - \frac{B_N^T}{||B_N||^2} A_N \frac{B_N B_T^T}{||B_T||^2} | \frac{||B_T||^2 B_T B_N^T}{||B_T B_N^T||^2}
\end{aligned}
\tag{4.21}
$$

$$
K_N = B_N^T A_T \frac{B_T B_N^T}{||B_T B_N^T||^2} - \frac{B_N^T}{||B_N||^2} A_N \tag{4.22}
$$

From (4.20)-(4.22) we can observe that the equality between system (4.11) and (4.12) always has a solution that does not depend on the networks trainable parameters K_N and C_N.

Thus, the aFORCE algorithm performs without any compromise solution, for any type of target system. This is due to the fact that the two interdependent problems of dynamic and output representation fitting between the two systems are handled by three independent trainable variables: the state-feedback gain K_N, the input layer B_N and the readout C_N.

□

By having two trainable structures, each focusing on one individual problem (tuning the dynamics of the reservoir and tuning the output representation of the network), aFORCE improves the approximation capabilities of current RC structures.

When analyzing the architecture proposed by the aFORCE algorithm in Figure 4.2, one could argue that the reservoir is an equivalent to the main cortical area of unspecialized neurons that transmit lots of types of data. By attaching a feedback to this reservoir (coupling specialized neurons to the main area in a feedback loop), the whole dynamic of the cortex changes: the specialized neurons are able to guide the unspecialized neurons into performing certain tasks (e.g., object recognition, color recognition, depth analysis, etc.). By having several such structures, that have been observed through MRI scans [128] for different tasks, the brain is able to process several types of data without spending much energy into training and reshaping the whole cortex for each different task. The biological role of the readout can be viewed from the opposite perspective: the closed-loop reservoir is coupled to a final layer of specialized neurons that transform and assemble the internal information into a desired, final representation (e.g., thoughts, memories, etc.).

In view of this, the following section introduces the proposed algorithm in more detail, followed by a series of numerical experiments that benchmark the new approach against the state-of-the-art FORCE method [12].

4.4 A novel methodology for training state-feedback ESNs

This section introduces aFORCE as a novel algorithm for training state-feedback ESNs. Given the training input-output data, the first stage of the proposed algorithm is to train the state-feedback gain and input layer using the EKF estimation [20],[21],[22]. The next stage is to determine the optimal non-linear readout, assumed to be in the form of a polynomial, for a selected order o. For that, the target output vector is projected onto the basis of the extended state vector $\overline{\mathbf{x}}$ (terms up to the n-th polynomial order) of the reservoir. The selection of the optimal polynomial terms is done in a greedy fashion, using a standard OFR algorithm [23]. The final number of presynaptic neurons that form the optimal readout are selected as the smallest number of neurons that leads to the maximum prediction accuracy on the validation data set. If the results do not meet the approximation error requirements, the order o of the multi-variate polynomial should be increased and the OFR selection should be redone.

The following section presents the novel aFORCE algorithm.

4.4.1 aFORCE Learning for ESN with Trainable Input Layers and Trainable State-Feedback Gains

Consider the following single-input-single-output ESN with state-feedback:

$$\begin{aligned} \mathbf{x}(k+1) &= f\left(W\mathbf{x}(k) + W_{in}\mathbf{u}(k) - W_{in}W_{fb}\mathbf{x}(k)\right) \\ y(k) &= H(\mathbf{x}(k)) \end{aligned} \tag{4.23}$$

where $f : \mathbb{R}^n \to \mathbb{R}^n$ is the activation function, the input with $l \in \mathbb{N}$ delays $u : \mathbb{R} \to \mathbb{R}, \mathbf{u} : \mathbb{R} \to \mathbb{R}^l, \mathbf{u}(k) = \begin{bmatrix} u(k) & u(k-1) & \dots & u(k-l) \end{bmatrix}^T$, $\mathbf{x} : \mathbb{R} \to \mathbb{R}^n, \mathbf{x} = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix}^T$ the states of the network, $W \in \mathbb{R}^{n \times n}$ the weight matrix, $W_{in} \in \mathbb{R}^{n \times l}$ the input-to-reservoir connection matrix, $W_{fb} \in \mathbb{R}^{l \times n}$ the state-feedback matrix, the output functional $H : \mathbb{R}^n \to \mathbb{R}$, $y : \mathbb{R}^n \to \mathbb{R}$ the output and $n \in \mathbb{N}^*$ the number of neurons in the reservoir,.

We can write in a more compact from as:

$$y(k) = M(f(\mathbf{x}(k), \mathbf{u}(k), W_{fb}, W_{in}), H(\mathbf{x}(k), W_{out})) \tag{4.24}$$

where $M : \mathbb{R}^n \; to \mathbb{R}$ is the maps the input $\mathbf{u}$ to output y.

An example of a 3rd order multivariate in $\mathbf{x}$ polynomial readout equation:

$$\begin{aligned} H(\mathbf{x}) = &\sum_{i=1}^{n} w_{0i} x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} w_{1ij} x_i x_j + \sum_{i=1}^{n} w_{2i} x_i^2 + \\ &+ \sum_{i=1}^{n} \sum_{j=i+1}^{n} w_{3ij} x_i^2 x_j + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \sum_{k=j+1}^{n} w_{4ijk} x_i x_j x_k + \sum_{i=1}^{n} w_{5i} x_i^3 \end{aligned} \tag{4.25}$$

The goal is to learn the weights W_{out}, W_{in}, W_{fb} that minimize the empirical risk:

$$\underset{W_{in}, W_{fb}, W_{out}}{\arg\min} \frac{1}{m} \sum_{k=1}^{N} \left(z(k) - M(f(\mathbf{x}(k), \mathbf{u}(k), W_{fb}, W_{in}), H(\mathbf{x}(k), W_{out})) \right)^2 \tag{4.26}$$

given the input and output observations $\{u(1), ..., u(N)\}, \{z(1), ..., z(N)\}$ generated by a target dynamical system:

$$z(k) = T(\bar{f}(\bar{\mathbf{x}}(k), \mathbf{u}(k)), \bar{H}(\bar{\mathbf{x}}(k))) \tag{4.27}$$

where $\bar{H}$ represents the output target function, $\bar{f}$ is the state target function.

4.4.1.1 INITIALIZATION AND STATE-FEEDBACK, INPUT LAYER ESTIMATION

Step 1 - O($3n$): Extend ESN states with the input layer W_{in} and state-feedback gain W_{fb}:

$$\begin{aligned} &\overline{\mathbf{x}} :\rightarrow \mathbb{R}^{3n}, \overline{\mathbf{x}} = \begin{bmatrix} x_1 & x_2 & \dots & x_n & w_{in-1} & w_{in-2} & \dots & w_{in-n} & w_{fb-1} & w_{fb-2} & \dots & w_{fb-n} \end{bmatrix}^T \\ &\overline{W}_{in} \in \mathbb{R}^{3n}, \overline{W}_{in} = \begin{bmatrix} W_{in} \\ O_{2n} \end{bmatrix} \\ &\overline{W}_{fb} \in \mathbb{R}^{1 \times 3n}, \overline{W}_{fb} = \begin{bmatrix} W_{fb} & O_{1x2n} \end{bmatrix} \\ &\overline{W} \in \mathbb{R}^{3n \times 3n}, \overline{W} = \begin{bmatrix} W & O_{2n} \\ O_n & I_{2n} \end{bmatrix} \end{aligned} \tag{4.28}$$

when $l = 1$. We have presented this extension for simplicity, but it can be applied for all $l \in \mathbb{N}^*$.

Step 2 - O(n^2): Identify the feedback gain W_{fb} and input weights W_{in} using the EKF estimation method on the following equation:

$$\overline{\mathbf{x}}(k+1) = f\left(\overline{W}\overline{\mathbf{x}}(k) + \overline{W}_{in}\mathbf{u}(k+1) - \overline{W}_{in}\overline{W}_{fb}\overline{\mathbf{x}}(k) - \overline{W}_{in}\overline{\mathbf{x}}^T(k)\overline{W}_{fb}{}^T \right) \tag{4.29}$$

with $\overline{x}_i(k) = \begin{Bmatrix} f(\overline{x}_i(k-1)), i \leq n \\ \overline{x}_i(k-1), i > n \end{Bmatrix}$.

4.4.1.2 FITTING THE READOUT MAP H

The state selection error:

$$ERR_j^{(p)} = \frac{\left\langle \bar{x}_j^{\perp(p-1)}, y^* \right\rangle_{L^2}}{||\bar{x}_j^{\perp(p-1)}||_{L^2}^2 ||y^*||_{L^2}^2} \tag{4.30}$$

where y^* is the target output signal, $\bar{x}_j^{\perp(p-1)}$ is the orthogonalized state of the initial state $\bar{x}_j$ at step $p-1$.

Step 3 - O(1): Setting the order of the multivariate polynomial readout map H in $\mathbf{x}$ $o = 2$ and the minimum cross-validation error threshold $e_{\min} = 0.15$ for the stopping criteria.

Step 4 - O(1): Initialize the OFR algorithm:

Compute reservoir state vector $\mathbf{x} = \begin{bmatrix} x_1 & ... & x_n \end{bmatrix}$ using equation (4.23). Generate the candidate model terms $\{\bar{x}_1^{\perp(0)}, ..., \bar{x}_q^{\perp(0)}\}$ of the multivariate polynomial readout of order $o \in \mathbb{N}$ in $\mathbf{x}$, where $q \in \mathbb{N}$ is the total number of terms associated to the polynomial order o. Set the error tolerance $\rho = 10^{-6}$ and $p = 1$.

Step 5 - O($2(q-p)+N+1$): Perform OFR iteration:

$$\begin{aligned}
&\bar{x}_j^{\perp(p)} = \bar{x}_j^{\perp(p-1)} - \frac{\left\langle \bar{x}_j^{\perp(p-1)}, \bar{x}_{p-1}^{\perp} \right\rangle_{L^2}}{||\bar{x}_{p-1}^{\perp}||_{L^2}^2} \bar{x}_{p-1}^{\perp}, j \in \{1, ..., q\} \backslash L^{(p-1)} \\
&l_p = \underset{j \in \{1,...,q\} \backslash L^{(p-1)}}{\arg\max} \; ERR_j^{(p)}, L^{(p)} = L^{(p-1)} \cup \{l_p\} \\
&ERR_p = ERR_{l_p}^{(p)}, \bar{x}_p^{\perp} = \bar{x}_{l_p}^{\perp(0)} \\
&W^{(p)} = \begin{bmatrix} w_1^{(p)} & ... & w_p^{(p)} \end{bmatrix} = y^* \begin{bmatrix} \bar{x}_1^{\perp(0)} \\ ... \\ \bar{x}_p^{\perp(0)} \end{bmatrix}^{-1}, \hat{y}^{(p)} = \sum_{k=1}^{p} w_k^{(p)} \bar{x}_{l_k}^{\perp(0)}
\end{aligned} \tag{4.31}$$

Step 6 - O($2N$): Current model validation:

If $\frac{\sum_{i=1}^{N} \left(\hat{y}^{(p)}(i) - y^*(i)\right)^2}{\sum_{i=1}^{N} \left(y^*(i) - mean(y^*)\right)^2} \leq e$, $e = \frac{\sum_{i=1}^{N} \left(\hat{y}^{(p)}(i) - y^*(i)\right)^2}{\sum_{i=1}^{N} \left(y^*(i) - mean(y^*)\right)^2}$ select $p, W^{(p)}, L^{(p)}$.

Step 7 - O(p): Calculate model prediction error for the validation data set:

$ERR_t = \sum_{l=1}^{p} ERR_l$, if $1 - ERR_t \leq \rho$, $p = q$.

Increment p , if $p < q$, repeat step 5.

Step 8 O(o): Cross-validation of the polynomial order:

If $e_{\min} < e$ and $o \leq 5$, increment o and repeat step 4.

Because q is of the order n^3 for a 3rd order polynomial, the total complexity of the aFORCE

is in the order of $O(n^3)$.

Using non-linear activation functions for the neurons inside the reservoir improves the approximation capabilities of the FORCE and aFORCE algorithms, at the expense of increasing the numerical complexity in determining the feedback gain using the EKF, in the case of aFORCE. It is worth observing that there is no increase in computation complexity for determining the trainable parameters when using the FORCE algorithm. However, the following sections, which present some numerical results, show that using more complex neuron models increase the approximation performance gap even further between the two aforementioned algorithms.

4.4.2 Simulation framework

In this section, we consider three types of target systems, to test and compare the novel aFORCE algorithms fitting performance with the state-of-the-art FORCE method [12]: a linear dynamical system with a linear output function, a linear state target system representation coupled to a non-linear output and a complete non-linear system in the form of the fruitfly photoreceptor model presented in [129].

For the first example, in the form of a linear-linear system, synthetic input-output data is created by having unfiltered white-noise as the input signal, in order to completely explore the frequency spectrum (i.e., phase and magnitude) which will lead us and the reader to a better understanding of the functionality and differences between the proposed aFORCE method and the state-of-the-art FORCE algorithm.

In the second linear - non-linear case study, the output response of the target system is generated by a filtered white-noise signal. The filter that has been used is in the form of a 7-th order Butterworth low-pass filter that has the following parameters: $F_s = 1Hz; F_n = 0.5Hz; F_{co} = 0.1Hz; w_s = 0.32Hz; R_p = 3dB; R_s = 20dB$ where F_s, F_n, F_{co} are the sampling, Nyquist and cutoff frequencies, w_s is the stopband frequency and R_p, R_s are the passband and stopband ripples.

The input-output data for the fruitfly photoreceptor has been selected from paper [129]. The authors present the response function of the photoreceptor to 5 different light stimuli, from which, our experiment considers Level 2.

The proposed state-feedback ESN architecture (4.23) and aFORCE algorithm is compared to the state-of-the-art output-feedback ESN (4.1) trained using the FORCE algorithm. The input layer and reservoirs are fully connected and randomly initialized. A number of different reservoir dimensionalities are tested to understand how the approximation performances of both algorithms and architectures are affected by the number of neurons in the reservoir. Linear and non-linear reservoirs are considered for a complete overview of the differences between the two methods.

To test the noise robustness of both methodologies, consider $\chi : \mathbb{R}^* \to [0,1]$ normally distributed (0,1) additive white noise and the following 3 types of Signal-to-Noise Ratios (SNRs): $S = y(t)\chi^{-1}(t)$: noiseless ($S = \infty$), low noise levels ($S = 50$) and high noise levels ($S = 10$).

For the the first numerical example, consider both readouts to be linear, in order to observe the benefits of a having the reservoir coupled with a state-feedback gain compared to an output-feedback coupling. The dominant dynamics are captured by training the input layer - state-feedback gain pair and the readout is only used to fit the targets output representation. The readout of the standard output-feedback architecture struggles to mimic the target output as the readout module is also used in modifying the internal dynamics of the reservoir.

For the second example, the readout of the state-feedback gain ESN that is trained using the aFORCE methodology is a non-linear 3rd order polynomial, compared to the linear readout of the output-feedback gain ESN trained using FORCE. This differentiation allows the reader to observe all the benefits of the enhanced proposed architecture. Instead of expanding the dimensionality of the reservoir, in order to capture the non-linearities, using a non-linear, easy to train, polynomial readout shows that the performance of a reservoir coupled with a non-linear output layer offers better approximation accuracy whilst keeping the reservoir small, when compared to the standard high-dimensionality reservoir-linear readout architecture.

For the third experiment, the input layer is coupled with delays for better dynamical approximation results. Both readouts that are trained using the aFORCE and FORCE algorithms are non-linear 3rd order polynomials in order to highlight the impact of non-linear components in capturing the dynamics of high non-linear tasks. As in the first simulation, having the same structures underlines the pivotal change made in swapping the type of feedback for better dynamical approximation that ultimately leads to better fitting results. The Wiener linear - non-linear system identification toolbox from MATLAB is considered as a benchmark in the case of linear reservoirs.

Moreover, changing the eigen-spectrum radius of the reservoir (i.e., $R \in (0,1], |\lambda_W| \leq R \leq 1$) depending on the dynamical evolution speed of the target system can represent another method that can dramatically affect the approximation capabilities of ESNs [18],[108]. The latter is best observed in the first example, where, the target has been chosen to have very small eigenvalues ($|\lambda| \leq 0.2$), whereas the initial reservoirs have been chosen to have eigenvalues on the verge of the unitary circle ($0.8 \leq |\lambda_W| \leq 0.9$).

The performance of both algorithms is measured using the Normalised Mean Squared Error (NMSE) given by:

$$NMSE = \frac{\sum_{i=1}^{M} (y(i) - y^*(i))^2}{\sum_{i=1}^{M} (y^*(i) - \bar{y^*})^2} \tag{4.32}$$

where $\bar{y^*} = \frac{1}{M} \sum_{i=1}^{M} y^*(i)$.

4.4.3 Synthetic linear - linear systems approximation

Consider the random stable linear discrete target systems of the following form:

$$\begin{aligned} \mathbf{x}(k+1) &= \mathbf{A}\mathbf{x}(k) + \mathbf{B}\mathbf{u}(k) \\ y(k) &= \mathbf{C}\mathbf{x}(k) + \chi(k) \end{aligned} \tag{4.33}$$

where $T = 1, k \in \mathbb{N}^*$ are the sampling time and sampling periods, $\mathbf{A} \in \mathbb{R}^{40\times40}, \mathbf{B} \in \mathbb{R}^{40\times1}, \mathbf{C} \in \mathbb{R}^{1\times40}, \mathbf{x} : \mathbb{N}^* \to \mathbb{R}^{40\times1}$ are the system matrix, input matrix, output matrix and state vector, $\mathbf{u} \in \mathbb{R}^* \to (0,1)$ is the uniformly distributed white noise input and $y \in \mathbb{N}^* \to \mathbb{R}$ is the output.

The eigenvalues of the system described in (4.33) are chosen to be very small, namely, $0 < |\lambda_{\mathbf{A}}| \leq 0.2$ which indicate very fast dynamics.

To illustrate the importance of having a state-feedback gain coupled to the reservoir of the ESN compared to a randomly initialized output-feedback gain, consider the initial reservoir to have slow evolving dynamics. Namely, in this experiment, eigenvalues that are close to the unitary circle: $0.8 \leq |\lambda_W| \leq 0.9$ are considered.

Figure 4.3 depicts the input-output representation of (4.33) and an average response of aFORCE and FORCE methodologies.

Dimension		**SNR**	**NMSE**	
Target	**Network**		**aFORCE**	**FORCE**
40	20	∞	0.0489	0.4473
		50	0.0512	0.4516
		10	0.0532	0.4920
	40	∞	0.0817	0.5280
		50	0.0867	0.2589
		10	0.1025	0.5203
	80	∞	0.0669	0.1484
		50	0.0505	0.2329
		10	0.0301	0.1805

Table 4.1: **NMSE Comparison between the aFORCE and the FORCE methodologies** of different reservoir dimensionalities, with respect to the linear-linear target system dimensionality and different SNRs

This numerical study has been specifically designed to show that when the eigenvalues of the target system are far from the initial reservoirs eigenvalues, increasing the reservoirs dimensionality of a fixed feedback structure [108] can not outperform a structure that includes an adapting feedback gain. Figure 4.3 c. shows that the output fitting of the target system (black) using the aFORCE method (red) is far more accurate than using the FORCE method (blue).

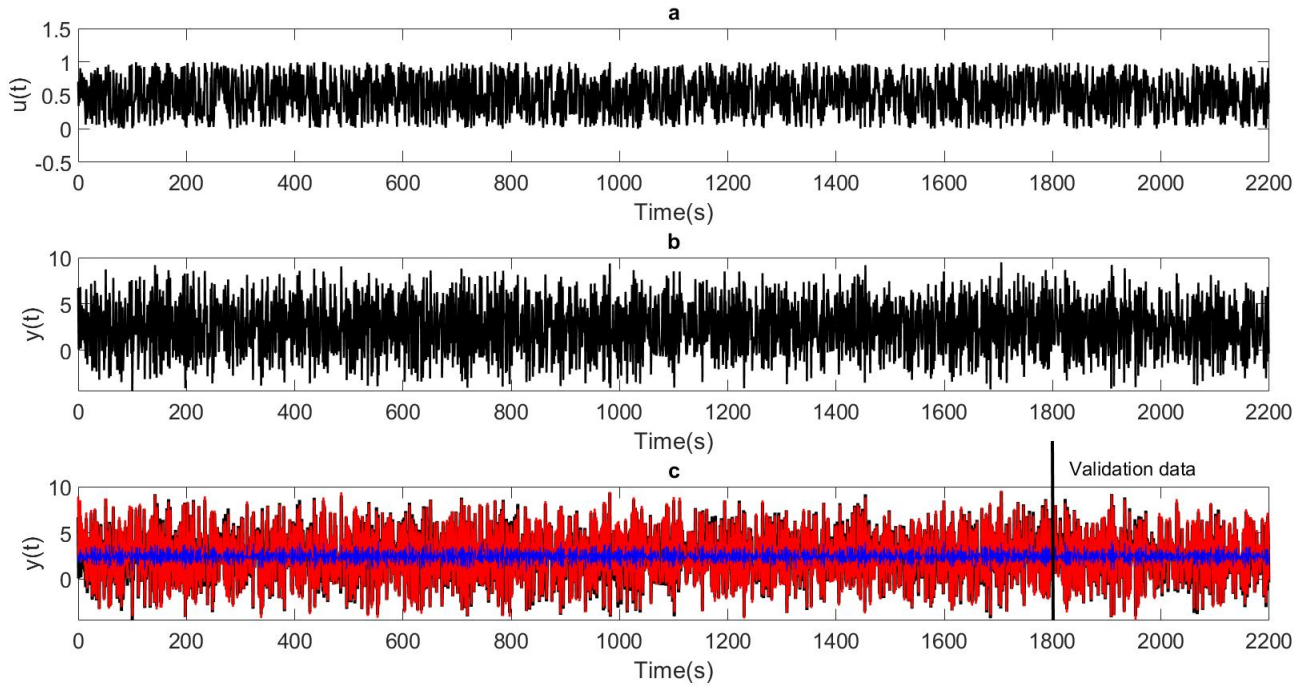

Figure 4.3: **ESN response to a synthetic linear-linear system :** a) uniformly distributed (0,1) white noise input signal b) response of the linear-linear target system used to fit the ESN model c) aFORCE ESN model predicted mean output (red) superimposed on the simulated estimation data (black) and on the FORCE ESN model predicted mean output (blue) for a linear reservoir with $N = 40$ neurons, maximum input lags $l = 1$, 1 output lag and $SNR = 10$ with aFORCE NMSE = 0.10, FORCE NMSE = 0.52 carried out on 100 tests

The results in Table 4.1 show that the aFORCE algorithm outperforms, on average, the state-of-the-art FORCE method. This suggests that the readout trained using the FORCE method cannot fit the output, whilst, at the same time, change the dynamics of the reservoir via the fixed output-feedback gain. This highlights the major advantage of the two separate trainable structures (i.e., input layer and state-feedback gain) exploited by the aFORCE.

The results in Table 4.1 also depict that the proposed algorithm is far less sensitive to the reservoirs dimensionality. It can be observed from Table 4.1 that the increase in performance from $N = 20$ to $N = 80$ is about 20% for the aFORCE, whilst the FORCE performs, on average 50%, better when the reservoir consists of more neurons. For this particular example, both algorithms are robust to different noise levels, as the difference in results for the same dimensionality, with different SNRs are negligible.

Analysing the Bode plot from Figure 4.4 shows that the ESN architecture trained using the FORCE method (blue) struggles to match the phase and magnitude of the target system (black),

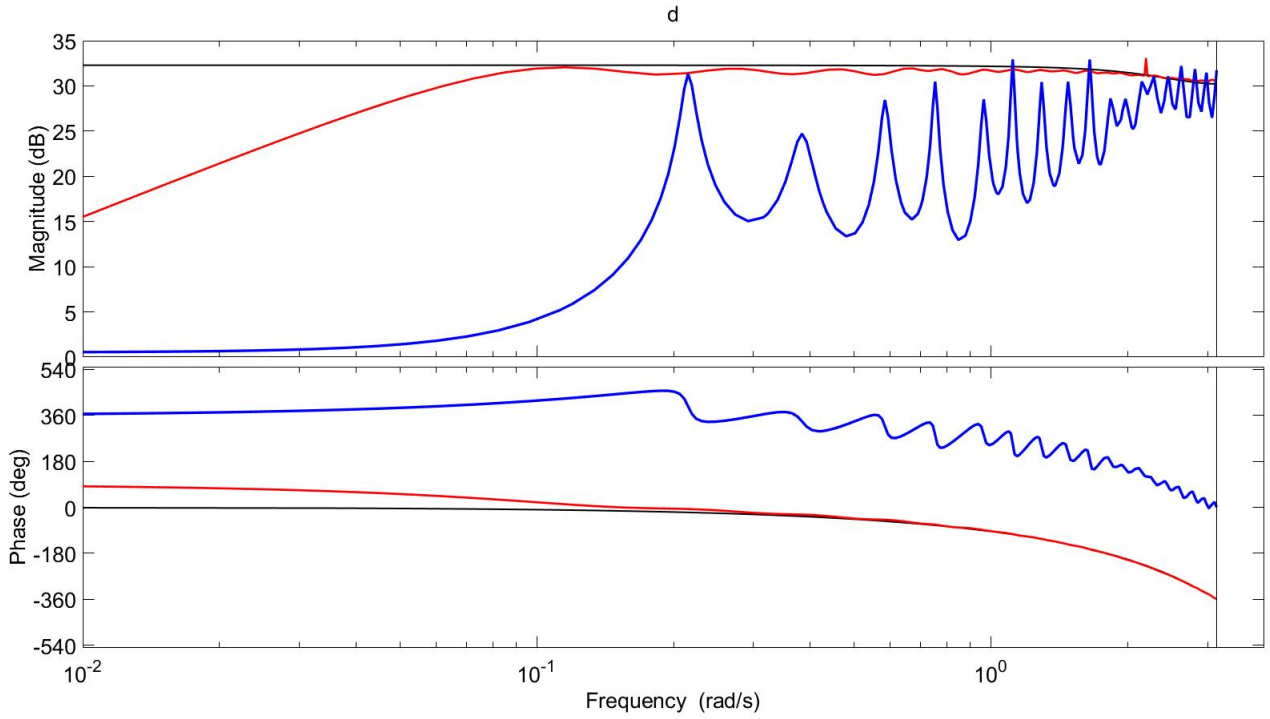

Figure 4.4: **Bode plot comparison:** aFORCE state-feedback gain ESN model (red) superimposed on the synthetic target system (black) and on the FORCE output-feedback gain ESN model (blue)

whilst the ESN trained using the aFORCE (red) matches the behavior of the target (black). This is to be expected as the EKF works perfectly for approximating linear systems such as the target chosen for this example, as defined in equation (4.33).

The next section introduces the second example that looks at the advantages that a non-linear polynomial readout has over a linear output structure.

4.4.4 Synthetic linear - non-linear systems approximation

Consider the randomly initialized stable linear - non-linear discrete target systems of the following form:

$$\begin{aligned} \mathbf{x}(k+1) &= \mathbf{A}\mathbf{x}(k) + \mathbf{B}\mathbf{u}(k) \\ y(k) &= \sum_{i=1}^{39} [0.5(x_{i+1} - x_i^2)^2 + (0.5 - x_i)^2] + \chi(k) \end{aligned} \tag{4.34}$$

where $T = 1, k \in \mathbb{N}^*$ are the sampling time and sampling periods, $\mathbf{A} \in \mathbb{R}^{40\times40}, \mathbf{B} \in \mathbb{R}^{40\times1}, \mathbf{x} : \mathbb{N}^* \rightarrow \mathbb{R}^{40\times1}$ are the system matrix, input matrix and state vector, $\mathbf{u} \in \mathbb{R}^* \rightarrow (0,1)$ is the uniformly distributed white noise input and $y \in \mathbb{N}^* \rightarrow \mathbb{R}$ is the output.

Figure 4.5 depicts the Input-Output representation of equation (4.34) and an average response of aFORCE and FORCE methodologies.

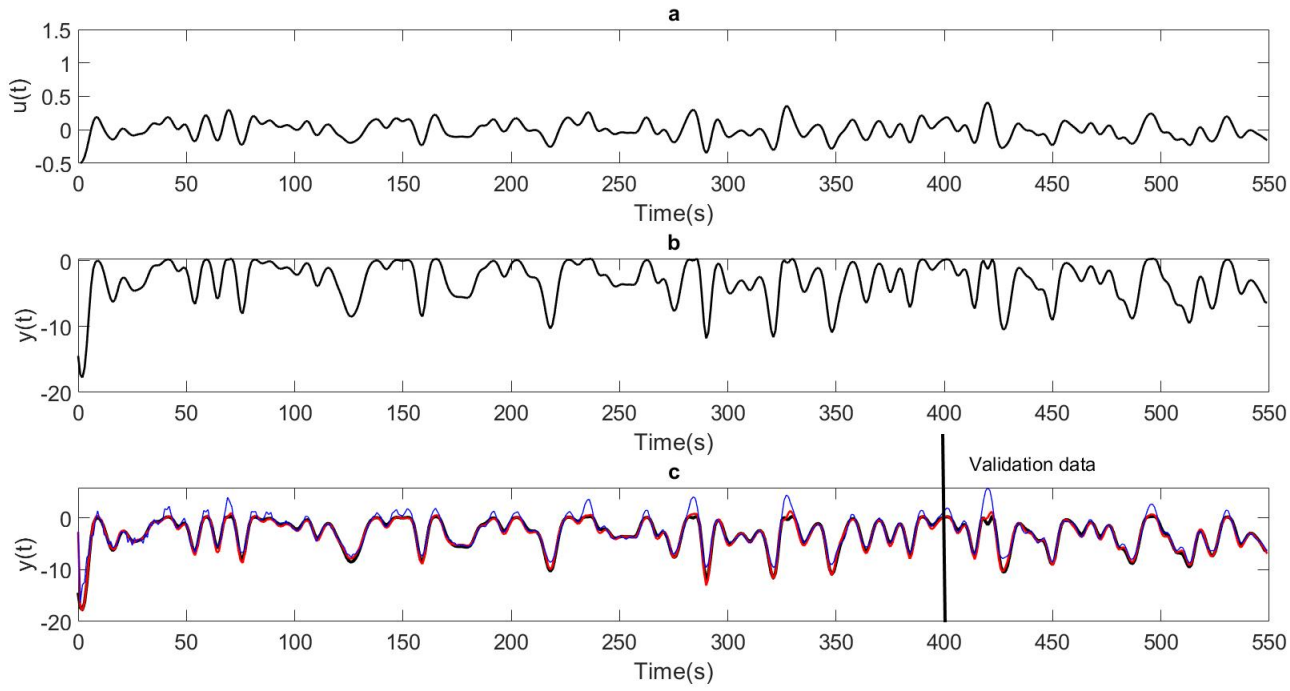

Figure 4.5: **ESN response to a synthetic linear - non-linear system:** a) uniformly distributed (0,1) filtered white noise input signal b) response of the linear - non-linear target system used to fit the ESN model c) aFORCE ESN model predicted mean output (red) superimposed on the simulated estimation data (black) and on the FORCE ESN model predicted mean output (blue) for a linear reservoir with $N = 80$ neurons, maximum input lags $l = 1$, 1 output lag and $SNR = 50$ with NMSE = 0.05 carried out on 100 tests

This numerical study has been specifically designed to show that for a linear - non-linear

target system, a linear-linear ESN with uniformly spread eigenspectrum cannot capture the behavior of a non-linear output representation with the same accuracy as a non-linear output layer. Moreover, the observations made in the previous section regarding the sepparation of the two main problems of dynamical and output fitting still hold. Figure 4.5 c. shows that the output fitting using the aFORCE approach is more accurate than the FORCE method.

Dimension		**SNR**	**NMSE**	
Target	**Network**		**aFORCE**	**FORCE**
40	20	∞	0.0392	0.6513
		50	0.0433	0.6490
		10	0.2721	0.6240
	40	∞	0.0142	0.4997
		50	0.0148	0.4021
		10	0.2722	0.4759
	80	∞	0.0601	0.2159
		50	0.0705	0.2162
		10	0.1799	0.2664

Table 4.2: **NMSE Comparison between the aFORCE and the FORCE methodologies** of reservoir dimensionalities, with respect to the linear - non-linear target system dimensionality and different SNRs

The results in Table 4.2 show that the aFORCE algorithm outperforms, on average, the state-of-the-art FORCE method. This is due to the fact that the linear readout of the FORCE ESN cannot capture the important non-linearities exhibited by the output function of the target system. In contrast, the polynomial readout of the aFORCE ESN, coupled with the trainable state-feedback gain and input layer prove to be superior in terms of approximation error with regards to reservoir dimensionality.

The results Table 4.2 also depict that the proposed algorithm is far less sensitive to the reservoirs dimensionality and noise. It can be observed from Table 4.2 that the increase in performance from $N = 20$ to $N = 80$ is about 50% for the aFORCE, whilst the FORCE performs, on average 70%, better when the reservoir contains more neurons.

Furthermore, aFORCE is more robust to different noise regimes. Interestingly, the standard FORCE architecture performs better under noisy conditions, compared to noiseless, for low dimension reservoirs.

The next section presents the last example using real data from the response of a fruitfly photoreceptor to light stimuli.

4.4.5 Fruitfly non-linear photoreceptor approximation

The data used in this study has been recorded in vivo from the photoreceptor of a fruitfly in the presence of different intensity light stimulus [129]. The light stimuli have been sampled at 2 kHz that have been low-pass filtered by elliptic filters with a 500 Hz cut-off frequency. The authors in [129] use five different intensities that were concatenated, in order to resemble light fluctuations that are present in a flies natural habitat. The light pattern has been selected as a naturalistic time-series of intensities specific to the Van Hateren natural stimuli collection. For this particular example, level 2 (L2) has been selected.

Figure 4.6 illustrates the input and output response of the fruitfly photoreceptor using the L2 stimuli, as well as the average response of a state-feedback gain ESN, using linear and non-linear reservoirs:

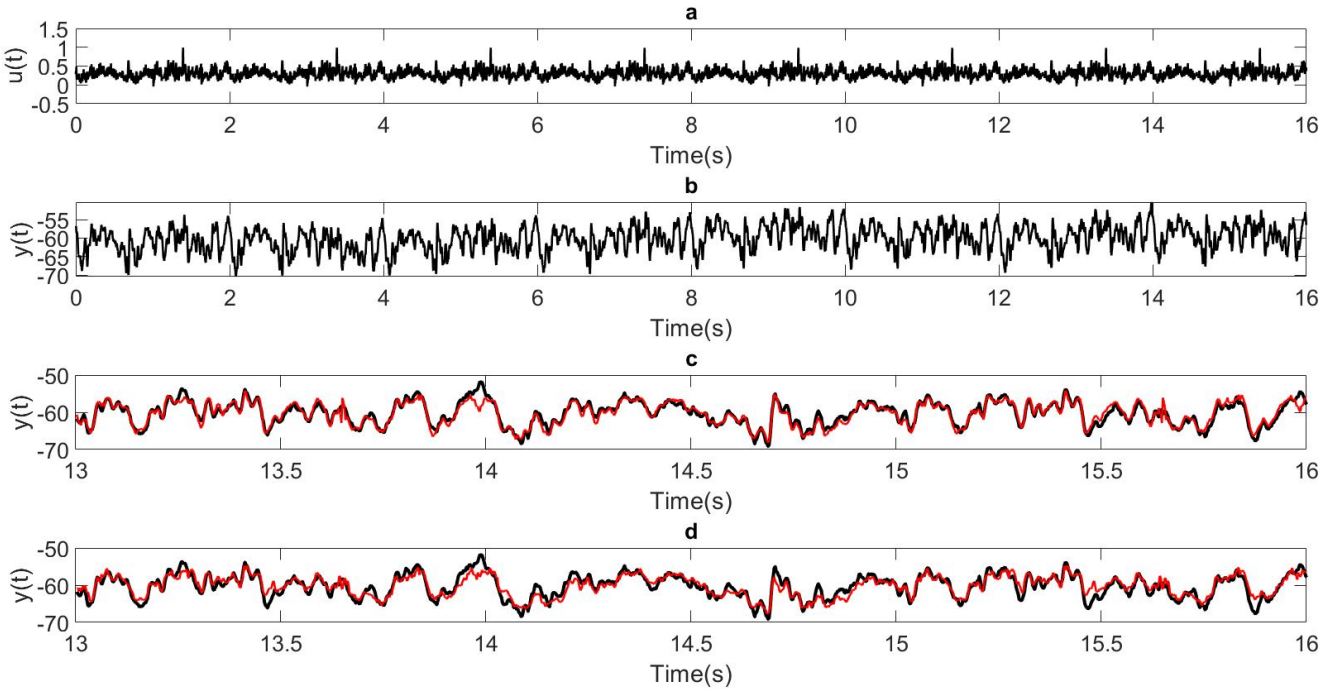

Figure 4.6: **ESN response to real-life data recorded from the photoreceptor of a Fruit Fly:** a) naturalistic stimuli b) in vivo photoreceptor responses used to fit the ESN model c) aFORCE ESN model predicted output (red) superimposed on the validation data extracted from the experimental photoreceptor response (black) for a hyperbolic tangent reservoir with $N = 11$ neurons, maximum input lags $l = 5$ and 1 output lag with NMSE = 0.12 d) ESN model predicted output (red) superimposed on experimental photoreceptor responses (black) for a linear reservoir with $N = 13$ neurons, maximum input lags $l = 7$ and 1 output lag with NMSE = 0.17

The original signal that has been recorded experimentally from the fruitflies photoreceptor consists of 2 second representative naturalistic time-series of intensities signal sampled initially at a frequency of 500 Hz. The data has then been preprocessed for analysis and experimental purposes to 400 Hz, has been concatenated eight times and the first 200 points representing the transitory regime have been eliminated, as suggested in [129] for single level characterization to ensure that the photoreceptor response is stationary. The eight newly created periods are split as follows: the first six periods are considered for training and model cross-validation. Each period has been sub sectioned as follows: from each period, the first 600 points are used for training purposes and are concatenated into a 3600-point training signal. The last 200 points from each period are concatenated into an 1200-point signal that is used to cross-validate and select the best model that is obtained from the OFR algorithm. The last two periods are used to test the final model.

The resulted values for the NMSE of both algorithms for a linear ESN with non-linear readout modules and the control identification method are presented in Tables 4.3, 4.4 and 4.5.

	aFORCE NMSE						
Dimensionality/Delay	**1**	**2**	**3**	**4**	**5**	**6**	**7**
11	0.28	0.26	0.23	0.21	0.19	0.18	0.17
12	0.33	0.26	0.23	0.21	0.19	0.18	0.16
13	0.28	0.23	0.22	0.23	0.24	0.21	0.18
14	0.30	0.26	0.21	0.20	0.19	0.18	0.18

Table 4.3: **aFORCE NMSE** with regards to different reservoir dimensionalities, different number of input delays and 1 output lag

	FORCE NMSE						
Dimensionality/Delay	**1**	**2**	**3**	**4**	**5**	**6**	**7**
11	0.25	0.30	0.26	0.28	0.26	0.25	0.26
12	0.26	0.25	0.24	0.26	0.24	0.25	0.24
13	0.25	0.24	0.25	0.26	0.26	0.25	0.24
14	0.27	0.24	0.23	0.23	0.23	0.25	0.23

Table 4.4: **FORCE NMSE** with regards to different reservoir dimensionalities, different number of input delays and 1 output lag

The results in Tables 4.3, 4.4 and 4.5 show that the state-of-the-art method has an overall lower accuracy compared to the proposed method. Introducing input delays did improve the accuracy of the FORCE method, but it could not change the dynamics of the reservoir to approximate the target tasks dynamics, compared to the aFORCE. When taking into consideration the variation of the structural parameters of system dimensionality and the number of input delays, the Wiener method outperforms the aFORCE algorithm for some cases (e.g., dimensionality 11, delay 7).

	Wiener NMSE						
Dimensionality/Delay	**1**	**2**	**3**	**4**	**5**	**6**	**7**
11	0.13	0.48	0.25	0.70	0.16	0.15	0.13
12	0.95	0.12	0.13	0.16	0.13	0.14	0.17
13	0.13	0.12	0.12	0.16	0.17	0.13	0.19
14	0.14	0.13	0.15	0.17	0.59	0.15	0.21

Table 4.5: **Wiener NMSE** with regards to different reservoir dimensionalities, different number of input delays and 1 output lag

Using a non-linear reservoir, namely, the hyperbolic tangent function, reveals that the approximation results could be further improved and proves the completeness of the proposed algorithm as it can be extended to non-linear, invertible reservoirs. The resulted values for the NMSE of both methodologies for non-linear differentiable ESNs coupled with non-linear readout modules are presented in Tables 4.6 and 4.7.

	aFORCE NMSE						
Dimensionality/Delay	**1**	**2**	**3**	**4**	**5**	**6**	**7**
11	0.28	0.29	0.22	0.20	0.19	0.18	0.16
12	0.29	0.25	0.22	0.20	0.19	0.18	0.17
13	0.27	0.24	0.22	0.20	0.19	0.18	0.17
14	0.29	0.25	0.22	0.21	0.19	0.18	0.17

Table 4.6: **aFORCE NMSE** with regards to different reservoir dimensionalities, different number of input delays and 1 output lag

.

	FORCE NMSE						
Dimensionality/Delay	**1**	**2**	**3**	**4**	**5**	**6**	**7**
11	0.46	0.38	0.56	0.34	0.36	0.47	0.47
12	0.28	0.35	0.40	0.36	0.28	0.27	0.32
13	0.45	0.57	0.43	0.31	0.25	0.20	0.25
14	0.78	0.75	0.33	0.38	0.25	0.21	0.23

Table 4.7: **FORCE NMSE** with regards to different reservoir dimensionalities, different number of input delays and 1 output lag - with red is highlighted the cases where FORCE performs better than aFORCE

The results in Tables 4.6 and 4.7 show that the state-of-the-art method has an overall lower accuracy compared to the proposed method. Introducing input delays did improve the accuracy of the FORCE method, but it could not change the dynamics of the reservoir to approximate the target tasks dynamics, compared to the aFORCE. Furthermore, changing the type of reservoir to

non-linear does not have a significant impact on the performance of the FORCE method when compared to the results presented in table 4.4. The only case when FORCE slightly outperforms aFORCE is for a reservoir of 12 neurons with 1 input delay.

4.5 Conclusions

This chapter proposed a new training algorithm that uses an alternative training procedure for computing the input layer and state-feedback gain via an EKF method [20],[21],[22] whilst training a non-linear readout module via an OFR algorithm [32]. The overall linear - non-linear structure allows learning of non-linear dynamics. The selection of the state-feedback gain such that, the dynamics of the network closer mimic the dynamics of the target system, enhances the overall approximation of a desired output target. Moreover, having a trainable state-feedback enforces the stability of the closed-loop network, compared to a randomly initialized output-feedback.

Compared to one-step prediction models, this algorithm falls under the category of model prediction, which offers better dynamical approximation results with the advantage of having access to more than one prediction step for the tuned model.

The new theoretical results proposed in Sections 4.2 and 4.3 indicate that the proposed aFORCE algorithm outperforms the standard readout training methods coupled with fixed, randomly initialized output gains, such as the FORCE algorithm [12]. This is due to the fact that a trainable separate state-feedback gain is able to model the closed-loop dynamics to be close to the dynamics of the unknown target system whilst the trainable readout module is only tasked to fit the target output. Numerical simulations carried out with synthetic and real data confirmed the theoretical findings and have showed that our proposed algorithm leads to much smaller reservoirs for similar approximation performance and better approximation results for higher reservoir sizes when compared to the dimensionalities of a target system. Moreover, the simulations highlight that the non-linear reservoir aFORCE performs better than the linear aFORCE. The numerical results show that the state-feedback plays a pivotal role in improving the approximation results.

Our study considered filtered and unfiltered data, simulated or collected in the real world with different levels of noise. Each study highlights different aspects in which the newly introduced aFORCE outperforms the standard FORCE method.

In all scenarios, the numerical results shown by the proposed aFORCE method outperform the ones shown by the state-of-the-art FORCE method. Moreover, our proposed method shows that the reservoir dimensionality can be reduced without a significant impact of the approximation capability, as a smaller reservoir is still able to capture the dominant dynamics of a given target system. Evaluating different levels of noise highlights the robustness of both methods.

It is worth noting that around less than 5% of the total possible connections between the reservoir and readout are required and that fully connected readouts achieve less accuracy on dynamical approximation tasks. This suggests that besides decoding stimulus features from the

evoked neuron activity, the new training method could be also used to characterize the functional specificity of neurons in the reservoir.

Due to the fact that in some cases, an increase in reservoir dimensionality does not translate into an increase in approximation performance, selecting the dominant neurons with regards to the dynamics of a target task would further improve the proposed algorithm. Along this line, the following chapter introduces a selection algorithm for the input layer and state-feedback gain parameters.

Chapter 5

State-feedback connectivity optimization of state-feedback ESNs

5.1 Introduction

Following the conclusions from Chapter 4, this chapter focuses on improving the performance of the aFORCE algorithm by optimizing the connectivity of the input layer and state-feedback gain with respect to the estimated dynamics of the target task.

To that end, let us reintroduce the basic concept of ESNs which consist of a dynamic reservoir in the form of an RNN, usually with sigmoid neural activation functions, having fixed, randomly assigned synaptic weights. The neurons map the inputs $u : \mathbb{R} \to \mathbb{R}$ to a n-dimensional state-space $\mathbf{x} : \mathbb{R} \to \mathbb{R}^n$ whilst the feedforward readout, having adjustable synaptic weights, maps the reservoir state $\mathbf{x}$ to the network output $y : \mathbb{R}^n \to \mathbb{R}$. Typically, an output-feedback gain that connects the output of the network with the input layer is attached to maximize the approximation capabilities of the ESN [9],[12].

The introduction of a fixed output or state-feedback gain [14] shapes the dynamics of the system via the eigenvalues of the closed-loop system which helps in bringing the networks output

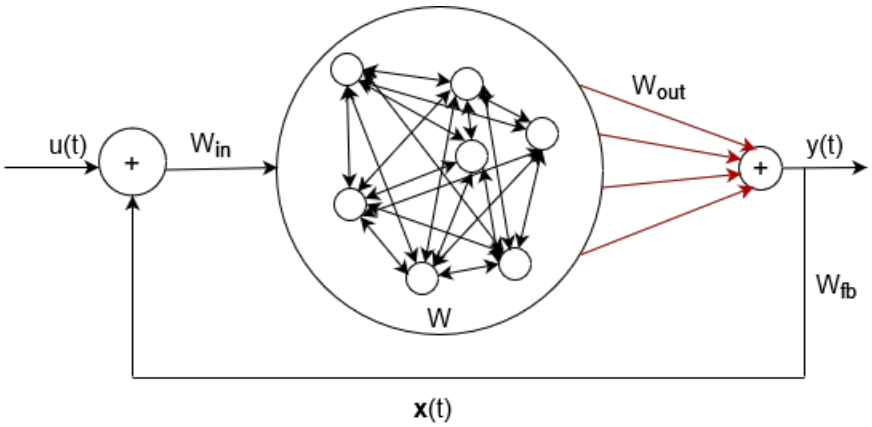

Figure 5.1: **Geeneric ESN architectures. The red arrows represent the parameters of the network that are trained**

closer to the target output. Optimizing the readout weights alone does not guarantee that these often competing goals can be achieved. In theory, matching the eigenvalue structure and the output map with the equivalent components of the target dynamical system would require optimizing the feedback gains as well as the readout weights.

In Chapter 4 we have introduced a novel training algorithm (aFORCE) which optimizes the input layer and state-feedback gain via an EKF procedure [20],[21],[22] in the first instance, and then, selects only the relevant parameters of the readout and computes their optimal value via an OFR algorithm [32]. This succession of steps, proved both theoretically and experimentally, produces models that significantly outperform randomly initialized output/state-feedback ESNs in which only the readout weights are trained [8],[9],[12].

The experiments conducted in Chapter 4 have shown that increasing the dimensionality of the reservoir does not necessarily translate into an increase in approximation performance. Hence, although fully trained state-feedback gains modify the dynamics of the reservoir, the readout consists of less than 5% of the total possible connections between the readout and the reservoir. This indicates that only some neurons dictate the dominant and most relevant dynamics of the given reservoir with regards to a specific target task.

In this chapter, we improve on the previously introduced aFORCE algorithm for state-feedback ESNs by reducing the number of trainable parameters in the input layer and state-feedback gain via an Orthogonal Forward Selection method which selects only the most relevant neurons with regards to the target output. To that end, we consider the classical ESN architecture [109] consisting of a linear dynamic reservoir and a non-linear readout (polynomial) map. The algorithm starts by selecting the dominant neurons and then train their corresponding parameters of the input layer and state-feedback gain. This is done in a similar fashion to the the original aFORCE, by applying an EKF estimation method as proposed in [20],[21],[22]. Note that the training is done only for

the selected dominant neurons. Finally, the readout training is done using the greedy optimization strategy, in the form of OFR [32] to identify the best synaptic connectivity for the readout module.

In the following sections of this chapter, the reader will be presented with the novel selection and training algorithm for state-feedback gain ESNs, some numerical studies and concluding remarks. Specifically, the following section presents partial state-feedback ESNs.

5.2 Partial state-feedback ESN

Consider the alternative ESN architecture having a variable dimension input layer and state-feedback gain, illustrated in Figure 5.2:

$$\begin{aligned} \mathbf{x}(k+1) &= f\left((W - W_{in}W_{fb})\mathbf{x}(k) + W_{in}\mathbf{u}(k)\right) \\ y(k) &= H(\mathbf{x}(k), W_{out}) \end{aligned} \tag{5.1}$$

where $\mathbf{u} = [u_1, ..., u_m]^T, \mathbf{x}, \mathbf{y}$ denote the input, state and output vectors, $f : \mathbb{R}^n \to \mathbb{R}^n$ is the reservoir activation function, $W \in \mathbb{R}^{n \times n}$ is the reservoir connection matrix, $W_{in} \in \mathbb{R}^{n \times m}$ is the input-to-reservoir connection matrix, $W_{fb} \in \mathbb{R}^{n \times p}$ is the output-feedback matrix and $H : \mathbb{R}^n \to \mathbb{R}^p$ is the output (readout) function with $W_{out} \in \mathbb{R}^{p \times n}$ the output weight matrix and $N_{red} \leqslant N$ is the number of non-zero trainable components in the state-feedback gain and input layer.

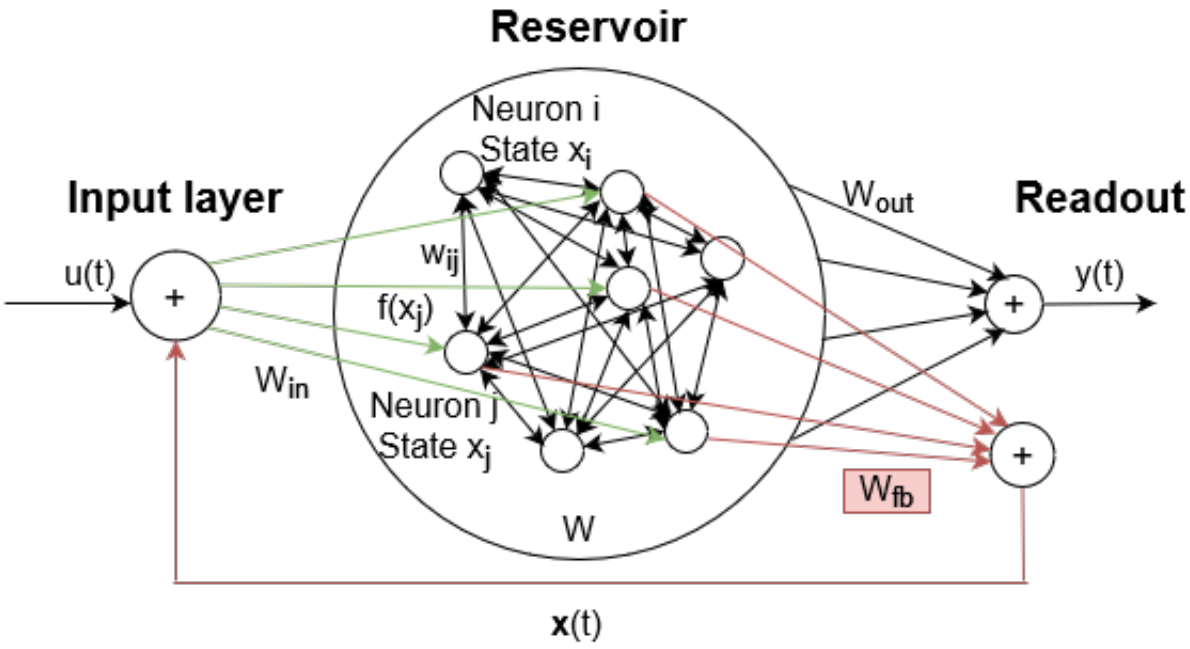

Figure 5.2: **Network architecture, connection selection and adjustable weights** for aFORCE-redux training algorithm

Note that equation (5.1) is similar equation (4.23). In this context, the newly introduced aFORCE algorithm starts the first training phase using an Orthogonal Forward Selection (OFS) algorithm to determine the most relevant nodes of the network with respect to the target task. After that, an EKF algorithm [20],[21],[22] is employed to control the magnitude of the training error with respect to the corresponding state-feedback gain and input layer parameters of the selected

dominant neurons. During this phase, selecting the optimal number of trainable parameters (in the form of the maximum number of trainable neurons) is also addressed. The output weights are then computed using an OFR algorithm [23]. During the second phase of the algorithm, the number of input lags and the polynomial readout model structure are selected.

Using a selection algorithm for identifying only the most relevant neurons, the aFORCE-redux algorithm avoids overfitting and reduces the computational costs associated with the previous aFORCE algorithm presented in Chapter 4.

From the proof of the original aFORCE algorithm presented in Section 4.3 regarding the state-feedback gain solution (4.22), it is likely that the inverse of B_N does not exist, because of the presence of multiple zeros that correspond to the neurons that have not been selected. To avoid this, a small noisy vector can be added in the form:

$$B_N = B_N + \delta, \delta : \mathbb{R} \to \mathbb{R}^{n \times l}, \delta \to 0 \tag{5.2}$$

In view of this, the following section introduces the proposed algorithm in more detail, followed by a series of numerical experiments that benchmark the new approach against the original aFORCE algorithm presented in Chapter 4.

5.3 A novel state-feedback gain dimensionality reduction and selection ESN training algorithm

This section introduces aFORCE-redux as a novel parameter dimensionality reduction and selection algorithm for training state-feedback ESNs. Given the training input-output data, the first stage of the algorithm is to identify the most relevant neurons with respect to the target task, and train the corresponding state-feedback gain and input layer parameters using the EKF estimation [20],[21],[22] for a selected number of trainable neurons N_{red}. The selection of the relevant nodes is done in a greedy fashion, using standard OFS. If the results do not meet the requirements, N_{red} is increased, the additional neuron is selected via OFS and the EKF procedure is redone. The next stage is to determine the optimal non-linear readout, assumed to be in the form of a polynomial, for a selected order o. The selection of the optimal polynomial terms is done in a Greedy fashion, using a standard OFR algorithm [23]. The final number of presynaptic neurons that form the optimal readout are selected as the smallest number of neurons that leads to the maximum prediction accuracy on the validation data set. If the results do not meet the approximation error requirements, the order o of the multi-variate polynomial should be increased and the OFR selection should be redone.

5.3.1 Algorithm presentation

Consider the following single-input-single-output ESN with state-feedback:

$$\begin{aligned} \mathbf{x}(k+1) &= f\left(W\mathbf{x}(k) + W_{in}\mathbf{u}(k+1) - W_{in}W_{fb}\mathbf{x}(k)\right) \\ y(k) &= H(\mathbf{x}(k)) \end{aligned} \tag{5.3}$$

where $n \in \mathbb{N}^*$ is the number of neurons in the reservoir, $f : \mathbb{R}^n \to \mathbb{R}^n$ the activation function, $u : \mathbb{R} \to \mathbb{R}, \mathbf{u} : \mathbb{R} \to \mathbb{R}^l, \mathbf{u}(k) = \begin{bmatrix} u(k) & u(k-1) & ... & u(k-l) \end{bmatrix}^T$ the input with $l \in \mathbb{N}$ delays, $\mathbf{x} : \mathbb{R} \to \mathbb{R}^n, \mathbf{x} = \begin{bmatrix} x_1 & x_2 & ... & x_n \end{bmatrix}^T$ the states of the network, $W \in \mathbb{R}^{n \times n}$ the weight matrix, $W_{in} \in \mathbb{R}^{n \times l}$ the input-to-reservoir connection matrix, $W_{fb} \in \mathbb{R}^{l \times n}$ the state-feedback matrix, the output functional $H : \mathbb{R}^n \to \mathbb{R}$, $y : \mathbb{R} \to \mathbb{R}$ the output and $N_{red} \leqslant N$ is the number of non-zero trainable components in the state-feedback gain and input layer.

We can write in a more compact from as:

$$y(k) = M(f(\mathbf{x}(k), \mathbf{u}(k), W_{fb}, W_{in}), H(\mathbf{x}(k), W_{out})) \tag{5.4}$$

where $M : \mathbb{R}^n \to \mathbb{R}$ is the mapping function that maps the input $\mathbf{u}$ to output y.

An example of a 3rd order multivariate in $\mathbf{x}$ polynomial readout equation:

$$\begin{aligned} H(\mathbf{x}) = &\sum_{i=1}^{n} w_{0i}x_i + \sum_{i=1}^{n}\sum_{j=i+1}^{n} w_{1ij}x_i x_j + \sum_{i=1}^{n} w_{2i}x_i^2 + \\ &+ \sum_{i=1}^{n}\sum_{j=i+1}^{n} w_{3ij}x_i^2 x_j + \sum_{i=1}^{n}\sum_{j=i+1}^{n}\sum_{k=j+1}^{n} w_{4ijk}x_i x_j x_k + \sum_{i=1}^{n} w_{5i}x_i^3 \end{aligned} \tag{5.5}$$

The goal is to learn the weights W_{out}, W_{in}, W_{fb} that minimize the empirical risk:

$$\underset{W_{in}, W_{fb}, W_{out}}{\arg\min} \frac{1}{m}\sum_{k=1}^{N} \left(z(k) - M(f(\mathbf{x}(k), \mathbf{u}(k), W_{fb}, W_{in}), H(\mathbf{x}(k), W_{out}))\right)^2 \tag{5.6}$$

given input and output observations $\{u(1), ..., u(N)\}, \{z(1), ..., z(N)\}$ generated by a target dynamical system:

$$z(k) = T(\bar{f}(\bar{\mathbf{x}}(k), \mathbf{u}(k)), \bar{H}(\bar{\mathbf{x}}(k))) \tag{5.7}$$

where $\bar{H}$ represents the output target function, $\bar{f}$ is the state target function.

5.3.1.1 RESERVOIR STATE SELECTION

The state selection error for determining the optimal state-feedback gain states is given by:

$$ERRS_j^{(p)} = \frac{\left\langle \bar{x}_j^{\perp(p-1)}, y^* \right\rangle_{L^2}}{||\bar{x}_j^{\perp(p-1)}||_{L^2}^2 ||y^*||_{L^2}^2} \tag{5.8}$$

where y^* is the target output signal, $\bar{x}_j^{\perp(p-1)}$ is the orthogonalized state of the initial state $\bar{x}_j$ at step $p-1$.

Step 1: Initialize the OFS algorithm:

Set the error tolerance $\rho = 10^{-6}$, $p = 1$, choose the number of trainable parameters $N_{red} \leqslant n$ and set cross-validation error threshold $e_{smin} = 0.1$ for the stopping criteria.

Step 2: Select optimal reservoir states corresponding to the target output:

$$\begin{aligned}
&\bar{x}_j^{\perp(p)} = \bar{x}_j^{\perp(p-1)} - \frac{\left\langle \bar{x}_j^{\perp(p-1)}, \bar{x}_{p-1}^{\perp} \right\rangle_{L^2}}{||\bar{x}_{p-1}^{\perp}||_{L^2}^2} \bar{x}_{p-1}^{\perp}, j \in \{1,...,q\} \backslash LS^{(p-1)} \\
&l_p = \underset{j \in \{1,...,q\} \backslash LS^{(p-1)}}{\arg\max} ERRS_j^{(p)}, L^{(p)} = LS^{(p-1)} \cup \{l_p\} \\
&ERRS_p = ERRS_{l_p}^{(p)}, \bar{x}_p^{\perp} = \bar{x}_{l_p}^{\perp(0)}
\end{aligned} \tag{5.9}$$

Increment p , if $p < N_{red}$, repeat step 2.

5.3.1.2 INITIALIZATION, STATE-FEEDBACK AND INPUT LAYER ESTIMATION

Step 3: Extend ESN states with the input layer W_{in} and state-feedback gain W_{fb}:

$$\begin{aligned}
&\overline{\mathbf{x}} :\rightarrow \mathbb{R}^{n+2N_{red}}, \\
&\overline{\mathbf{x}} = \begin{bmatrix} x_1 & x_2 & \dots & x_n & w_{in-1} & w_{in-2} & \dots & w_{in-N_{red}} & w_{fb-1} & w_{fb-2} & \dots & w_{fb-N_{red}} \end{bmatrix}^T \\
&\overline{W}_{in} \in \mathbb{R}^{n+2N_{red}}, \overline{W}_{in} = \begin{bmatrix} W_{in} \\ O_{n+N_{red}} \end{bmatrix} \\
&\overline{W}_{fb} \in \mathbb{R}^{1 \times 2n+N_{red}}, \overline{W}_{fb} = \begin{bmatrix} W_{fb} & O_{1x(n+N_{red})} \end{bmatrix} \\
&\overline{W} \in \mathbb{R}^{n+(2N_{red}) \times 3n}, \overline{W} = \begin{bmatrix} W & O_{2N_{red}} \\ O_n & I_{2N_{red}} \end{bmatrix}
\end{aligned} \tag{5.10}$$

when $l = 1$. We have presented this extension for simplicity, but it can be applied for all $l \in \mathbb{N}^*$.

Step 4: Identify the feedback gain W_{fb} and input weights W_{in} using the EKF estimation method on the following equation:

$$\overline{\mathbf{x}}(k+1) = f\left(\overline{W}\overline{\mathbf{x}}(k) + \overline{W}_{in}\mathbf{u}(k+1) - \overline{W}_{in}\overline{W}_{fb}\overline{\mathbf{x}}(k) - \overline{W}_{in}\overline{\mathbf{x}}^{T}(k)\overline{W}_{fb}{}^{T}\right) \tag{5.11}$$

with $\overline{x}_i(k) = \left\{ \begin{array}{c} f(\overline{x}_i(k-1)), i \leq n \\ \overline{x}_i(k-1), i > n \end{array} \right\}$ and fixed readout $W_{out} = [11...1]$.

Step 5: Cross-validation of the number of trainable parameters:

If $e_{smin} < e$, $e = \frac{\sum_{i=1}^{M} (W_{out}\overline{\mathbf{x}} - y^*(i))^2}{\sum_{i=1}^{M} (y^*(i) - mean(y^*))^2}$ and $N_{red} \leq n$, increment N_{red} and repeat step 2.

5.3.1.3 FITTING THE READOUT MAP H

The state selection error for determining the optimal output layer states is given by:

$$ERR_j^{(p)} = \frac{\left\langle \bar{x}_j^{\perp(p-1)}, y^* \right\rangle_{L^2}}{||\bar{x}_j^{\perp(p-1)}||_{L^2}^2 ||y^*||_{L^2}^2} \tag{5.12}$$

where y^* is the target output signal, $\bar{x}_j^{\perp(p-1)}$ is the orthogonalized state of the initial state $\bar{x}_j$ at step $p-1$.

Step 6: Setting the order of the multivariate polynomial readout map H in $\mathbf{x}$ $o = 2$ and the minimum cross-validation error threshold $e_{\min} = 0.15$ for the stopping criteria.

Step 7: Initialize the OFR algorithm:

Compute reservoir state vector $\mathbf{x} = \left[\begin{array}{ccc} x_1 & ... & x_n \end{array} \right]$ using equation (5.3). Generate the candidate model terms $\{\bar{x}_1^{\perp(0)}, ..., \bar{x}_q^{\perp(0)}\}$ of the multivariate polynomial readout of order $o \in \mathbb{N}$ in $\mathbf{x}$, where $q \in \mathbb{N}$ is the total number of terms associated to the polynomial order o. Set the error tolerance $\rho = 10^{-6}$ and $p = 1$.

Step 8: Perform OFR iteration:

$$\begin{aligned}
&\bar{x}_j^{\perp}(p) = \bar{x}_j^{\perp}(p-1) - \frac{\left\langle \bar{x}_j^{\perp(p-1)}, \bar{x}_{p-1}^{\perp} \right\rangle_{L^2}}{||\bar{x}_{p-1}^{\perp}||_{L^2}^2} \bar{x}_{p-1}^{\perp}, j \in \{1,...,q\} \backslash L^{(p-1)} \\
&l_p = \underset{j \in \{1,...,q\} \backslash L^{(p-1)}}{\arg\max} ERR_j^{(p)}, L^{(p)} = L^{(p-1)} \cup \{l_p\} \\
&ERR_p = ERR_{l_p}^{(p)}, \bar{x}_p^{\perp} = \bar{x}_{l_p}^{\perp}(0) \\
&W^{(p)} = \begin{bmatrix} w_1^{(p)} & ... & w_p^{(p)} \end{bmatrix} = y^* \begin{bmatrix} \bar{x}_1^{\perp}(0) \\ ... \\ \bar{x}_p^{\perp}(0) \end{bmatrix}^{-1}, \hat{y}^{(p)} = \sum_{k=1}^{p} w_k^{(p)} \bar{x}_{l_k}^{\perp}(0)
\end{aligned} \tag{5.13}$$

Step 9: Current model validation:
If $\frac{\sum_{i=1}^{M} \left(\hat{y}^{(p)}(i) - y^*(i)\right)^2}{\sum_{i=1}^{M} \left(y^*(i) - mean(y^*)\right)^2} \leq e$, $e = \frac{\sum_{i=1}^{M} \left(\hat{y}^{(p)}(i) - y^*(i)\right)^2}{\sum_{i=1}^{M} \left(y^*(i) - mean(y^*)\right)^2}$ select $p, W^{(p)}, L^{(p)}$.
Step 10: Calculate model prediction error for the validation data set:
$ERR_t = \sum_{l=1}^{p} ERR_l$, if $1 - ERR_t \leq \rho$, $p = q$.
Increment p , if $p < q$, repeat step 8.
Step 11: Cross-validation of the polynomial order:
If $e_{\min} < e$ and $o \leq 5$, increment o and repeat step 7.

In the following sections, the considered numerical examples test and compare the novel aFORCE-redux algorithms fitting performance with the previously introduced aFORCE method: a linear dynamical system with a linear output function, a linear state target system representation coupled to a non-linear output and a complete non-linear system in the form of the fruitfly photoreceptor model presented in [129].

5.3.2 Synthetic linear - linear systems approximation

For the first example, in the form of a linear-linear system, synthetic input-output data is created by having unfiltered white-noise as the input signal, in order to completely explore the frequency spectrum (i.e., phase and magnitude) which will lead us and the reader to a better understanding of the functionality and differences between the proposed aFORCE-redux method and the original aFORCE algorithm. Both of the readouts trained by the aforementioned methodologies are linear.

Consider the random stable linear discrete target systems of the following form:

$$\begin{aligned} \mathbf{x}(k+1) &= \mathbf{A}\mathbf{x}(k) + \mathbf{B}\mathbf{u}(k) \\ y(k) &= \mathbf{C}\mathbf{x}(k) + \chi(k) \end{aligned} \tag{5.14}$$

where $T = 1, k \in \mathbb{N}^*$ are the sampling time and sampling periods, $\mathbf{A} \in \mathbb{R}^{40\times40}, \mathbf{B} \in \mathbb{R}^{40\times1}, \mathbf{C} \in \mathbb{R}^{1\times40}, \mathbf{x} : \mathbb{N}^* \to \mathbb{R}^{40\times1}$ are the system matrix, input matrix, output matrix and state vector, $\mathbf{u} \in \mathbb{R}^* \to (0,1)$ is the uniformly distributed white noise input, $y \in \mathbb{N}^* \to \mathbb{R}$ is the output, $0 < |\lambda_{\mathbf{A}}| \leq 0.2$, and the ESN eigenvalues $0.8 \leq |\lambda_W| \leq 0.9$.

Analysing the Bode plot from Figure 5.3 shows that the ESN architecture trained using the aFORCE-redux algorithm (blue) closely matches the phase and magnitude of the ESN architecture trained using the original aFORCE algorithm (red), the behavior of the original target (black) and an 8-dimensional truncated version (green). Furthermore, it is also revealed that the classic output-feedback ESN architecture trained using the FORCE learning method (cyan) [12] does not follow the same trajectory as the rest of the systems, which is also supported by the results in the Tables provided in Section 4.3.3.

The results in Tables 5.1 and 5.2 show that the aFORCE-redux algorithm outperforms, on average, the original aFORCE method for different numbers of selected trainable neurons (NG). A possible explanation can be consisted of the fact that selecting only the relevant neurons of the reservoir with respect to the specific task reduces overfitting when compared to fully trained ESNs. This is supported by the fact that for some instances, a larger reservoir with less trainable parameters performs better than a smaller reservoir with fully trained parameters (e.g., aFORCE-redux $N = 80, SNR = \text{inf}, NG = 40$ performs better than the aFORCE for $N = 40, SNR = \text{inf}$)

The results in Tables 5.1 and 5.2 also depict that optimizing the connectivity of the state-feedback gain and input layer are consistent with the observations made in Chapter 4, in the sense that aFORCE based algorithms are robust to reservoir dimensionality and noise.

The next section introduces the second example which analyses the fitting performance of the aFORCE-redux algorithm compared to the original aFORCE method with respect to a linear -

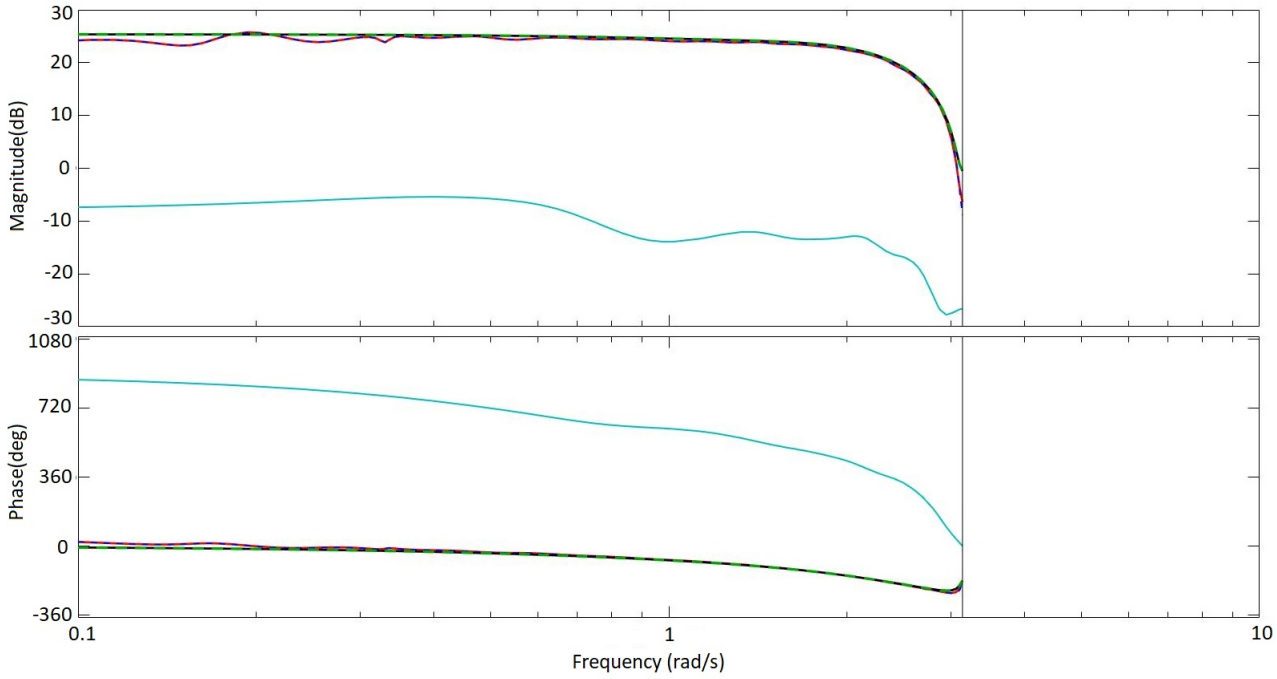

Figure 5.3: **Bode plot comparison:** ESN model aFORCE predicted bode (red) superimposed on the ESN model aFORCE-redux predicted bode (blue) superimposed on the simulated target system bode (black) superimposed on the truncated target system bode (green) superimposed on the output-feedback ESN FORCE predicted bode (cyan) for a linear reservoir having $N = 40$ neurons, maximum input lags $l = 1$, trained states $NG = 10$, 1 output lag, truncation 8 and $SNR = 50$ with $NMSE = 0.012$

Dimension		**SNR**	**NG**	**NMSE**	
Target	**Network**			**aFORCE**	**aFORCE-redux**
40	20	∞	10	0.0489	0.0250
		50	10	0.0512	0.1075
		10	10	0.0532	0.0071
	40	∞	10	0.0817	0.0432
		50	10	0.0867	0.0517
		10	10	0.1025	0.0138
	80	∞	10	0.0669	0.0021
		50	10	0.0505	0.0443
		10	10	0.0301	0.0174

Table 5.1: **NMSE Comparison between the aFORCE and the aFORCE-redux methodologies** of different reservoir dimensionalities, with respect to the linear-linear target system dimensionality and different SNRs

non-linear target system.

Dimension		SNR	NG	NMSE	
Target	**Network**			**aFORCE**	**aFORCE-redux**
40	20	∞	10	0.0489	0.0250
		50	10	0.0512	0.1075
		10	10	0.0532	0.0071
	40	∞	20	0.0817	0.0523
		50	20	0.0867	0.0741
		10	20	0.1025	0.0842
	80	∞	40	0.0669	2.7254e-06
		50	40	0.0505	0.0441
		10	40	0.0301	0.0191

Table 5.2: **NMSE Comparison between the aFORCE and the aFORCE-redux methodologies** of different reservoir dimensionalities, with respect to the linear-linear target system dimensionality and different SNRs

5.3.3 Synthetic linear - non-linear systems approximation

In this linear - non-linear case study, the output response of the target system is generated by a filtered white-noise signal. The filter that has been used is in the form of a 7-th order Butterworth low-pass filter that has the following parameters: $F_s = 1Hz; F_n = 0.5Hz; F_{co} = 0.1Hz; w_s = 0.32Hz; R_p = 3dB; R_s = 20dB$ where F_s, F_n, F_{co} are the sampling, Nyquist and cutoff frequencies, w_s is the stopband frequency and R_p, R_s are the passband and stopband ripples.

The readouts trained using the aFORCE and aFORCE-redux are non-linear 3rd order polynomials.

Consider the randomly initialized stable linear - non-linear discrete target systems of the following form:

$$\begin{aligned} \mathbf{x}(k+1) &= \mathbf{A}\mathbf{x}(k) + \mathbf{B}\mathbf{u}(k) \\ y(k) &= \sum_{i=1}^{39} [0.5(x_{i+1} - x_i^2)^2 + (0.5 - x_i)^2] + \chi(k) \end{aligned} \tag{5.15}$$

where $T = 1, k \in \mathbb{N}^*$ are the sampling time and sampling periods, $\mathbf{A} \in \mathbb{R}^{40\times40}, \mathbf{B} \in \mathbb{R}^{40\times1}, \mathbf{x} : \mathbb{N}^* \rightarrow \mathbb{R}^{40\times1}$ are the system matrix, input matrix and state vector, $\mathbf{u} \in \mathbb{R}^* \rightarrow (0,1)$ is the uniformly distributed white noise input and $y \in \mathbb{N}^* \rightarrow \mathbb{R}$ is the output.

Figure 5.4 depicts the Input-Output representation of equation (5.15) and an average response of the aFORCE-redux method.

Dimension		**SNR**	**NG**	**NMSE**	
Target	**Network**			**aFORCE**	**aFORCE-redux**
40	20	∞	10	0.0392	0.1015
		50	10	0.0433	0.0986
		10	10	0.2721	0.1951
	40	∞	10	0.0142	0.0641
		50	10	0.0148	0.1089
		10	10	0.2722	0.0842
	80	∞	10	0.0601	0.0912
		50	10	0.0705	0.0799
		10	10	0.1799	0.0742

Table 5.3: **NMSE Comparison between the aFORCE and the aFORCE-redux methodologies** of different reservoir dimensionalities, with respect to the linear - non-linear target system dimensionality and different SNRs

The results in Tables 5.3 and 5.4 show that increasing the number of trainable parameters (NG) did improve the approximation results of the aFORCE-redux algorithm, but there are cases where the novel proposed algorithm is outperformed by the original aFORCE. This suggests that training

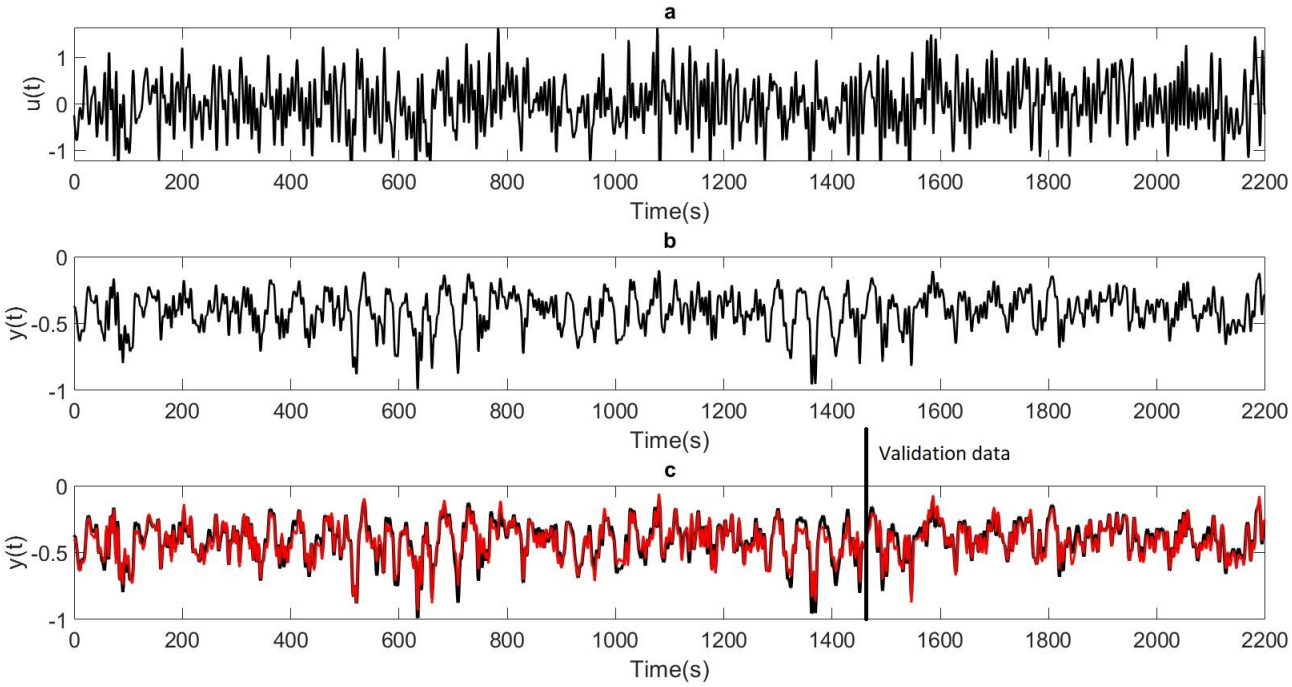

Figure 5.4: **ESN response to a synthetic linear - non-linear system:** a) uniformly distributed (0,1) filtered white noise input signal b) response of the linear - non-linear target system used to fit the ESN model c) aFORCE-redux ESN model predicted mean output (red) superimposed on the simulated estimation data (black) and on the FORCE ESN model predicted mean output (blue) for a linear reservoir with $N = 40$ neurons, maximum input lags $l = 1$, 1 output lag, $NG = 20$ and $SNR = 50$ with NMSE = 0.09 carried out on 100 tests

Dimension		**SNR**	**NG**	**NMSE**	
Target	**Network**			**aFORCE**	**aFORCE-redux**
40	20	∞	10	0.0392	0.1015
		50	10	0.0433	0.0986
		10	10	0.2721	0.1951
	40	∞	20	0.0142	0.0541
		50	20	0.0148	0.0589
		10	20	0.2722	0.0242
	80	∞	40	0.0601	0.0834
		50	40	0.0705	0.0510
		10	40	0.1799	0.0713

Table 5.4: **NMSE Comparison between the aFORCE and the aFORCE-redux methodologies** of different reservoir dimensionalities, with respect to the linear - non-linear target system dimensionality and different SNRs

the parameters corresponding to the unselected neurons can capture additional subtle dynamics of the target task. It can be observed from Tables 5.3 and 5.4 that for some instances, a larger reservoir with less trainable parameters performs better that a smaller reservoir with fully trained parameters (e.g., aFORCE-redux $N = 80, SNR = 10, NG = 40$ performs better than the aFORCE for $N = 40, SNR = 10$). It is interesting to highlight that, under heavy noise conditions, training less parameters offer better results when compared to training all the parameters.

The next section presents the last example which uses real data from the response of a fruitfly photoreceptor to light stimuli.

5.3.4 Fruitfly non-linear photoreceptor approximation

The data used in this study has been recorded in vivo from the photoreceptor of a fruitfly in the presence of different intensity light stimulus [129]. The light stimuli have been sampled at 2 kHz that have been low-pass filtered by elliptic filters with a 500 Hz cut-off frequency. The authors in [129] use five different intensities that were concatenated, in order to resemble light fluctuations that are present in a flies natural habitat. The light pattern has been selected as a naturalistic time-series of intensities specific to the Van Hateren natural stimuli collection. For this particular example, level 2 (L2) has been selected.

The input layer of the neural networks are coupled with delays for better dynamical approximation results. Both readouts that are trained using the aFORCE-redux and aFORCE algorithms are non-linear 3rd order polynomials in order to highlight the impact of non-linear components in capturing the dynamics of high non-linear tasks.

Figure 5.5 illustrates the input and output response of the fruitfly photoreceptor using the L2 stimuli:

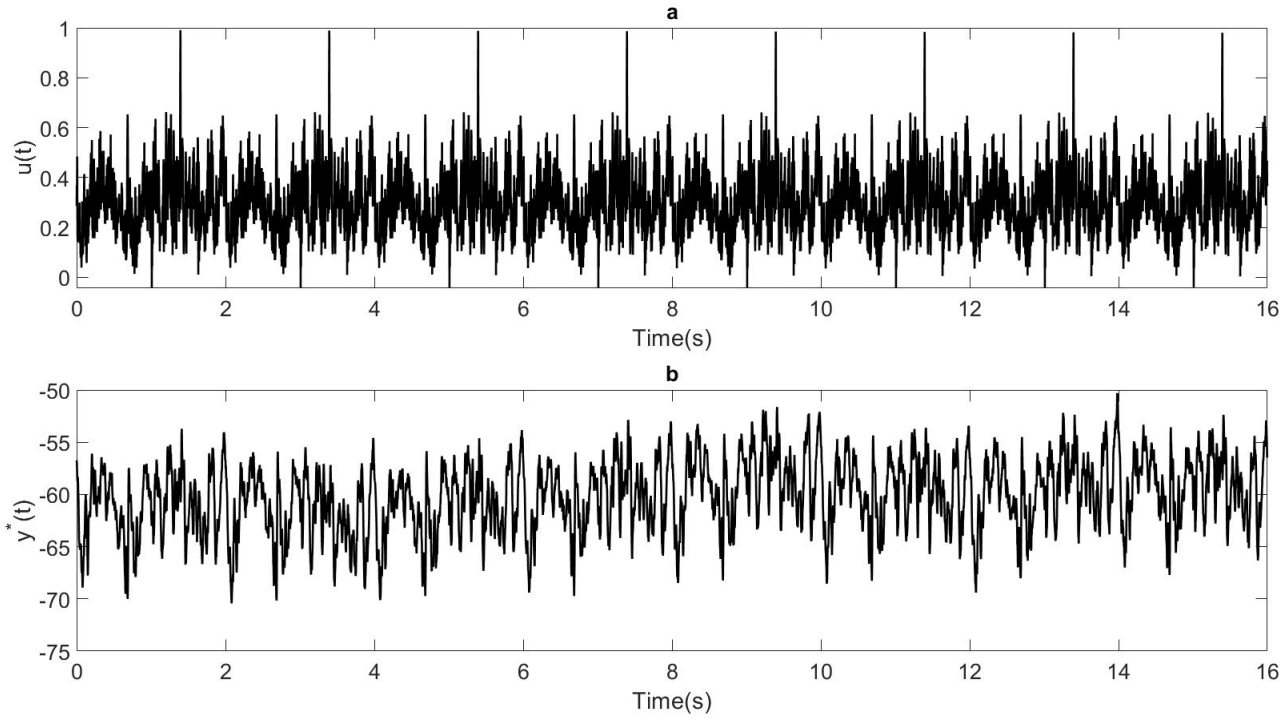

Figure 5.5: **Real-life data recorded from the photoreceptor of a Fruit Fly:** a) naturalistic stimuli b) in vivo photoreceptor responses used to fit the ESN model

The original signal that has been recorded experimentally from the fruitflies photoreceptor consists of 2 second representative naturalistic time-series of intensities signal sampled initially

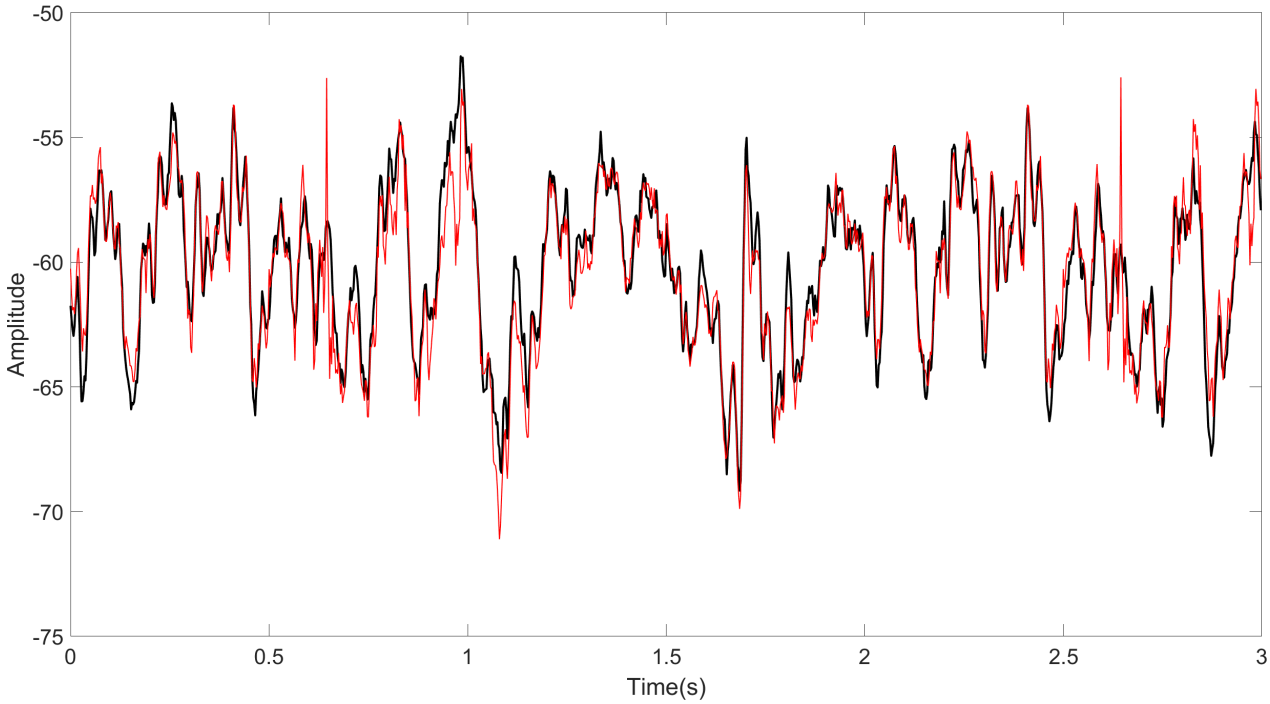

Figure 5.6: **ESN response to real-life data recorded from the photoreceptor of a Fruit Fly**: ESN model predicted output (red) superimposed on the experimental Fruitfly photoreceptor response (black) for a linear reservoir having $N = 13$ neurons, $l = 5$ input lags, $NG = 4$ states trained and 1 output lag with $NMSE = 0.12$

	aFORCE-redux NMSE						
Dimensionality/Delay	**1**	**2**	**3**	**4**	**5**	**6**	**7**
7	0.52	0.48	0.50	0.38	0.38	0.25	0.21
8	0.33	0.37	0.32	0.34	0.32	0.26	0.20
9	0.25	0.27	0.24	0.21	0.27	0.24	0.16
10	0.18	0.16	0.16	0.15	0.15	0.12	0.13
11	0.12	0.14	0.12	0.13	0.12	0.12	0.13
12	0.12	0.11	0.11	0.11	0.11	0.10	0.09
13	0.11	0.10	0.09	0.10	0.09	0.09	0.09

Table 5.5: **aFORCE-redux NMSE** with regards to different reservoir dimensionalities, different number of input delays and $NG = 4$ trained states for a linear ESN

at a frequency of 500 Hz. The data has then been preprocessed for analysis and experimental purposes to 400 Hz, has been concatenated eight times and the first 200 points representing the transitory regime have been eliminated, as suggested in [129] for single level characterization to ensure that the photoreceptor response is stationary. The eight newly created periods are split as follows: the first six periods are considered for training and model cross-validation. Each period has been sub sectioned as follows: from each period, the first 600 points are used for training purposes

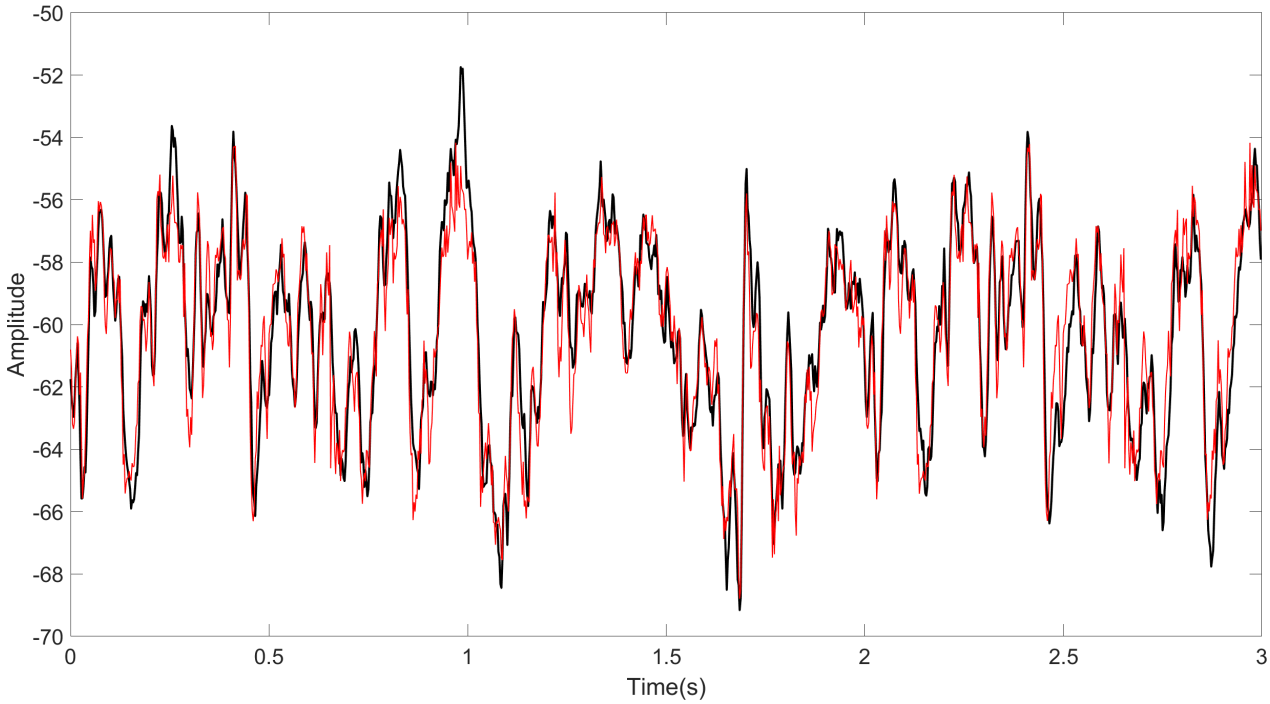

Figure 5.7: **ESN response to real-life data recorded from the photoreceptor of a Fruit Fly**: ESN model predicted output (red) superimposed on experimental photoreceptor responses (black) for a non-linear reservoir having $N = 13$ neurons, $l = 7$ input lags, $NG = 4$ states trained and 1 output lag with $NMSE = 0.13$

	aFORCE-redux NMSE						
Dimensionality/Delay	**1**	**2**	**3**	**4**	**5**	**6**	**7**
7	0.29	0.29	0.24	0.16	0.16	0.16	0.13
8	0.28	0.31	0.27	0.25	0.19	0.17	0.15
9	0.26	0.36	0.36	0.29	0.20	0.19	0.13
10	0.36	0.42	0.36	0.26	0.20	0.15	0.15
11	0.34	0.42	0.39	0.29	0.23	0.21	0.20
12	0.43	0.43	0.42	0.23	0.17	0.15	0.13
13	0.38	0.31	0.27	0.20	0.22	0.13	0.13

Table 5.6: **aFORCE-redux NMSE** with regards to different reservoir dimensionalities, different number of input delays and $NG = 4$ trained states for a non-linear ESN

and are concatenated into a 3600-point training signal. The last 200 points from each period are concatenated into an 1200-point signal that is used to cross-validate and select the best model that is obtained from the OFR algorithm. The last two periods are used to test the final model.

Comparing the results of the original aFORCE algorithm on the Fruifly data presented in Tables 4.3 and 4.6 with the results of the newly introduced aFORCE-redux presented in Tables 5.5 and 5.6, shows that the aFORCE-redux performs better, on average, compared to the original method.

A possible explanation is that selecting only the relevant neurons of the reservoir with respect to the specific task reduces overfitting when compared to fully trained ESNs. It is interesting to highlight that, compared to the original aFORCE method, where the non-linear reservoir performed better than the linear reservoir, for the newly proposed aFORCE-redux method, the linear reservoir performs better than the hyperbolic tangent one.

The next section presents the main conclusions of this chapter.

5.4 Conclusions

This chapter proposed a new training algorithm that improves on the aFORCE method proposed in Chapter 4. The aFORCE-redux is an alternative training procedure that selects only the relevant neurons that dictate the dynamics of the reservoir with regards to the target task and computes the input layer and state-feedback gain via an EKF method whilst training a non-linear readout module via an OFR algorithm. The advantages that have been highlighted in Chapter 4 are preserved, whilst reducing the number training parameters.

For benchmarking purposes, the same experiments conducted in Chapter 4 regarding filtered and unfiltered data, simulated or collected in the real world with different levels of noise were conducted.

Numerical simulations carried out with synthetic and real data have showed that the parameter reduction training algorithm leads to smaller reservoirs for similar approximation performance and better approximation results for higher reservoir sizes when compared to the dimensionalities of the target system and to the same reservoir trained with the original aFORCE. Moreover, the simulations highlight that the non-linear reservoir trained with the aFORCE-redux performs better than the linear reservoir trained with the aFORCE-redux for linear - linear and linear - non-linear tasks, whilst a linear reservoir performs better in the Fruifly experiment.

The numerical analysis of the novel aFORCE-redux method show that the number of trainable parameters can be reduced with negligible impact on the approximation capability as the dominant trained neurons are still able to capture the most important dynamics of a given target system. Comparing to the original aFORCE, shows that reducing the number of trainable parameters decreases the risk of overfitting. For instances, the aFORCE-redux performs slightly better than the original aFORCE in high noise conditions.

On the other hand, in some particular noise-free non-linear examples, the aFORCE performs better than the aFORCE-redux, which highlights the utility and importance of training all parameters when it comes to approximating highly complex tasks. In contrast, in all high noise scenarios across all simulation settings considered training less parameters is more efficient. For example, the aFORCE-redux outperforms the original aFORCE method.

One limitation of this approach is that running multiple tasks in parallel using the same reservoir is not feasible due to the trainable internal structures (state-feedback gain and input layer) directly modify the dynamics of the closed-loop. In this context, the next chapter introduces a novel switching state-feedback ESN architecture which offers increased approximation accuracy by taking advantage of the proposed aFORCE based methods.

Chapter 6

A novel architecture for ESNs by using switching input layers, state-feedbacks and readouts

6.1 Introduction

Given the limited applicability of the proposed aFORCE based methods to process multiple tasks in parallel, this chapter addresses this shortcoming by proposing a novel switching state-feedback ESN architecture. The novel structure offers increased approximation accuracy by taking advantage of the proposed aFORCE based methods.

To that end, let us reintroduce the basic concept of ESNs which consist of a dynamic reservoir in the form of an RNN, usually with sigmoid neural activation functions, having fixed, randomly assigned synaptic weights. The neurons map the inputs $u : \mathbb{R} \to \mathbb{R}$ to a n-dimensional state-space $\mathbf{x} : \mathbb{R} \to \mathbb{R}^n$ [14], whilst the feedforward readout, having adjustable synaptic weights, maps the reservoir state $\mathbf{x}$ to the network output $y : \mathbb{R}^n \to \mathbb{R}$. Typically, an output-feedback gain that connects the output of the network with the input layer is attached to maximize the approximation capabilities of the ESN [9],[12].

Current RC architectures [124] exploit efficient readout training methods [9],[17],[107],[123] that achieve comparable approximation results to that of fully trained RNNs whilst requiring minimal computational power. Moreover, the flexibility of only training the output layer allows RC networks to perform multiple tasks simultaneously by using the same input layer, output-feedback loop and reservoir whilst connecting additional trainable readouts in parallel.

In Chapters 4 and 5, we have analysed the limitations of closed-loop ESNs in the presence of an output-feedback gain with the role of altering the dynamics of the reservoir to closer match the dynamics of the unknown target system. Because the output-feedback gain is directly connected to the readout layer, training the readout and the output-feedback gain is unlikely to offer the optimal solution with regards to the two competing different problems: dynamical fitting of the reservoir and output fitting of the target signal. Often, the aforementioned problems have competing solutions (observed in [12] and proven in Section 4.2).

State-feedback RC architectures offer better numerical tractability compared to the classical output-feedback RC architectures [12], observed by Maass [14]. In Chapter 4, we have proven that it is necessary to change the output-feedback gain with a state-feedback gain to obtain better approximation results. Moreover, separating the two main problems of fitting the reservoir to match the dynamics of the target state via a state-feedback loop and output fitting of the target signal via a trainable readout eliminates the risk of the reservoir becoming unstable [9],[12],[14]. This also improves the approximation results as the two main training variables become independent. This is particularly useful as the two aforementioned problems often have competing solutions.

It is also worth preserving the advantages mentioned in Sections 4.5 and 5.4 of the aFORCE based methods by training the input layer and state-feedback and selecting the readout connectivity via OFR [23],[24].

Previous types of parallel computations consists of neural network ensembles [5],[41] and network committees [5],[130]. Instead of employing a single, larger network for single-task computing, several smaller networks are trained in parallel having their responses combined to obtain optimal task approximation. Frequently an ensemble of models performs better than any individual model, because the various errors of the models "average out." Ensemble averaging creates a group of networks, each with low bias and high variance, then combines them to a new network usually with low bias and low variance. This gives an obvious strategy: create a set of experts with low bias and high variance, and then average them. In other words, the main goal is to create a set of experts with varying parameters; frequently, these are the initial synaptic weights, although other factors (such as the learning rate, momentum etc.) may be varied as well. The resulting committee is almost always less complex than a single network that would achieve the same level of performance

Current generation parallel computing network architectures are in the form of RCs with

parallel trainable readouts [9],[12],[15]. Instead of employing multiple smaller and trainable networks that combine their response for single task approximation, this new approach prefers a randomly-initialized high-dimensional reservoir coupled with parallel trainable readouts used for performing multi-task computations.

The introduction of a trainable state-feedback gain for shaping the dynamics of the system and a trainable input layer for maximizing the approximation capabilities of the ESN, makes running several tasks in parallel unfeasible. This is due to the fact that each input layer and state-feedback gain is specifically designed for each target task and should be retrained when a new task is added or updated. Thus, we propose a novel switching Echo State Network (SESN) architecture to provide better approximation capabilities than current parallel structures.

In this chapter, we propose a new architecture using state-feedback ESNs that switch between input layers, state-feedback gains and readout modules for each individual task to produce models that significantly outperform the output-feedback ESNs in which only the parallel readout layers are trained for each task [8],[9],[12]. To that end, we adopt the ESN architecture [109] consisting of a linear dynamic reservoir and a non-linear (polynomial) readout map. Training the input layer - state-feedback gain - readout module pairs is implemented using the aFORCE methodology that has been presented in Chapter 4. For an in-depth analysis of the newly introduced architecture, we also analyse the impact and switching stability [131],[132] of non-linear differentiable reservoirs (e.g., hyperbolic tangent activation functions).

In the following sections of this chapter, the reader will be presented with the novel architecture, theoretical stability analysis, the additional steps required to implement the architecture based on the aFORCE algorithm, numerical study using real data from the fruitfly photoreceptor and concluding remarks. Specifically, the following section introduces the novel architecture.

6.2 Novel switching ESN architecture

Consider a discrete-time non-linear family of target systems described by the following equation [133]:

$$x_{k+1} = G_{\tau(k)}(x_k, u_k), k \in \mathbb{Z}_+ \tag{6.1}$$

with the output equation:

$$y_k = M_{\tau(k)}(x_k) \tag{6.2}$$

where $x_k \in \mathbb{R}^n$ is the state, $u_k \in \mathbb{R}^{d_u}$ is the input, $y_k \in \mathbb{R}^{d_y}$ is the output, $G_{\tau(k)} : \mathbb{R}^n \times \mathbb{R}^{d_u} \to \mathbb{R}^n$ are arbitrary non-linear state functions and, $H_{\tau(k)} : \mathbb{R}^n \to \mathbb{R}^{d_y}$ the output functions. The logical rule that governs the switching between the arbitrary non-linear functions $G_{\tau(k)}$ and output functions $M_{\tau(k)}$ is represented by a class of piecewise constant sequences $\tau : \mathbb{Z}^+ \to \mathscr{I}, \mathscr{I} = \{1, ..., K\}$.

Consider the state-feedback gain ESN architecture initially introduced in Chapter 4 and altered here by using a decoupled state-feedback gain as illustrated in Figure 4.2:

$$\begin{aligned} \mathbf{x}(k+1) &= f\left((W - W_{in_{fix}} W_{fb})\mathbf{x}(k) + W_{in}\mathbf{u}(k)\right) \\ y(k) &= H(\mathbf{x}(k), W_{out}) \end{aligned} \tag{6.3}$$

where $\mathbf{u} = [u_1, ..., u_m]^T, \mathbf{x}, \mathbf{y}$ denote the input, state and output vectors, $f : \mathbb{R}^n \to \mathbb{R}^n$ is the reservoir activation function, $W \in \mathbb{R}^{n \times n}$ is the reservoir connection matrix, $W_{in} \in \mathbb{R}^{n \times m}$ is the input-to-reservoir connection matrix, $W_{in_{fix}} = [11....1]^T$ an additional feedback gain, $W_{fb} \in \mathbb{R}^{n \times p}$ is the state-feedback matrix and $H : \mathbb{R}^n \to \mathbb{R}^p$ is the output (readout) function with $W_{out} \in \mathbb{R}^{p \times n}$ the output weight matrix.

Classical output-feedback ESN architectures [9],[12],[28],[123] use the same output-feedback gain to run multiple tasks in parallel, at the expense of approximation accuracy, as the dynamics of the reservoir are not specifically designed each individual task. It is proven in Chapters 4 and 5, that introducing and training a state-feedback gain, via the aFORCE and aFORCE-redux algorithms, to specifically tune the dynamics of the open-loop reservoir to each particular task. This proves to outperform state-of-the-art output-feedback gain ESN architectures and their specific training algorithms, such as the Echo State feedback clamping method [9] and the FORCE algorithm [12]. In order to preserve the advantages of the state-feedback ESN architecture and the aFORCE based training algorithms, some changes are required to facilitate running multiple tasks using the same reservoir. However, this can only be achieved in a serial fashion because of the state-feedback loop that alters the dynamics of the reservoir and of an input layer which further improves the approximation capabilities of the network.

Hence, let us consider the discrete-time switching ESN architecture given by:

$$\begin{aligned} \mathbf{x}(k+1) &= f\left((W - W_{in\,fix}W_{fb_{\sigma(k)}})\mathbf{x}(k) + W_{in\,\sigma(k)}\mathbf{u}(k)\right) \\ y(k) &= H_{\sigma(k)}(\mathbf{x}(k), W_{out\,\sigma(k)}) \end{aligned} \tag{6.4}$$

with the same notations as in equation (6.3). The logical rule that governs the switching between the $H \in \mathbb{N}$ closed-loop subsystems generated by the input layer, state-feedback gain and the output map is represented by a class of piecewise constant sequences $\sigma : \mathbb{Z}^+ \to \mathscr{I}, \mathscr{I} = \{1, ..., K\}$.

In [12], the authors proposed training the input layers of subnetworks included in the reservoir that can act like state-feedback gains for modeling the overall closed-loop dynamics of the network with respect to different target tasks. The input layer of the subnetwork is the only trainable component and the training is done the same way as for the readout module presented in Section 2.4.2.1. In this setting, Figure 6.1 depicts the exact architecture proposed in [12].

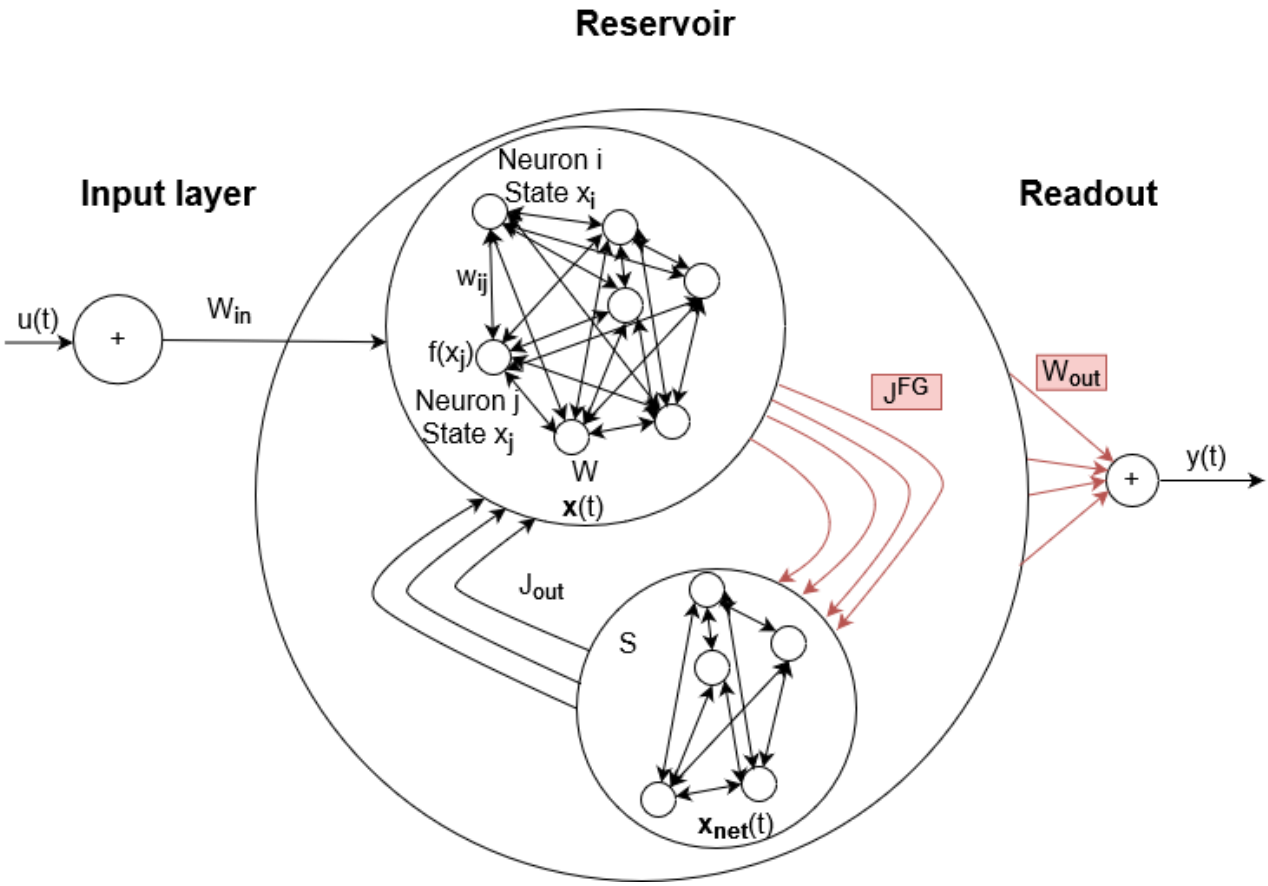

Figure 6.1: **Switching FORCE ESN architecture and adjustable weights**

Lemma 6.1 *Training the input layer J^{FG} of subnetwork S does not modify the dynamics of the reservoir.*

Proof. Consider the subnetwork equation given by:

$$\begin{aligned} \mathbf{x_{net}}(k+1) &= f\left(S\mathbf{x_{net}}(k) + J^{FG}\mathbf{x}(k)\right) \\ y(k) &= f(\mathbf{x_{net}}(k), J_{out}) \end{aligned} \tag{6.5}$$

where $\mathbf{x} \in \mathbb{R}^{\mathbf{n}}$ are the reservoirs states, $\mathbf{x_{net}} \in \mathbb{R}^{g}$ are the subnetwork states, $S \in \mathbb{R}^{g \times g}$ is the system matrix, $J^{FG} \in \mathbb{R}^{g \times n}$ the input layer, $J_{out} \in \mathbb{R}^{n \times g}$ is the output layer.

For simplicity, consider $f : \mathbb{R}^{g} \to \mathbb{R}^{g}, f = 1$. The equivalent transfer function of (6.5):

$$TF(s) = J_{out}(sI - S)^{-1} J^{FG} \tag{6.6}$$

Equation (6.6) shows that modifying J^{FG} does not change the eigenvalues of $sI - S$, which dictates the dynamics of system (6.5). This also shows that the subnetwork acts like a randomly initialized state-feedback gain. □

In other words, the overall state-feedback structure is still randomly initialized and the authors basically proposed a switching ESN structure with random feedback gains. The only special characteristic of the proposed architecture is that these specific neurons that form the subnetwork inside the reservoir do not receive direct inputs.

It is worth noting that switching systems require further stability guarantees, in addition to the stability of each subsystem [131]. Although each subsystem is stable, the resulting switching sequence can lead the overall system into instability. In the following section we explore different global asymptotic stability conditions are explored for both linear and non-linear reservoirs.

In this context, the following section introduces the required stability condition for both linear and non-linear reservoirs. The additional steps required to deploy aFORCE or aFORCE-redux in this setting are also introduced.

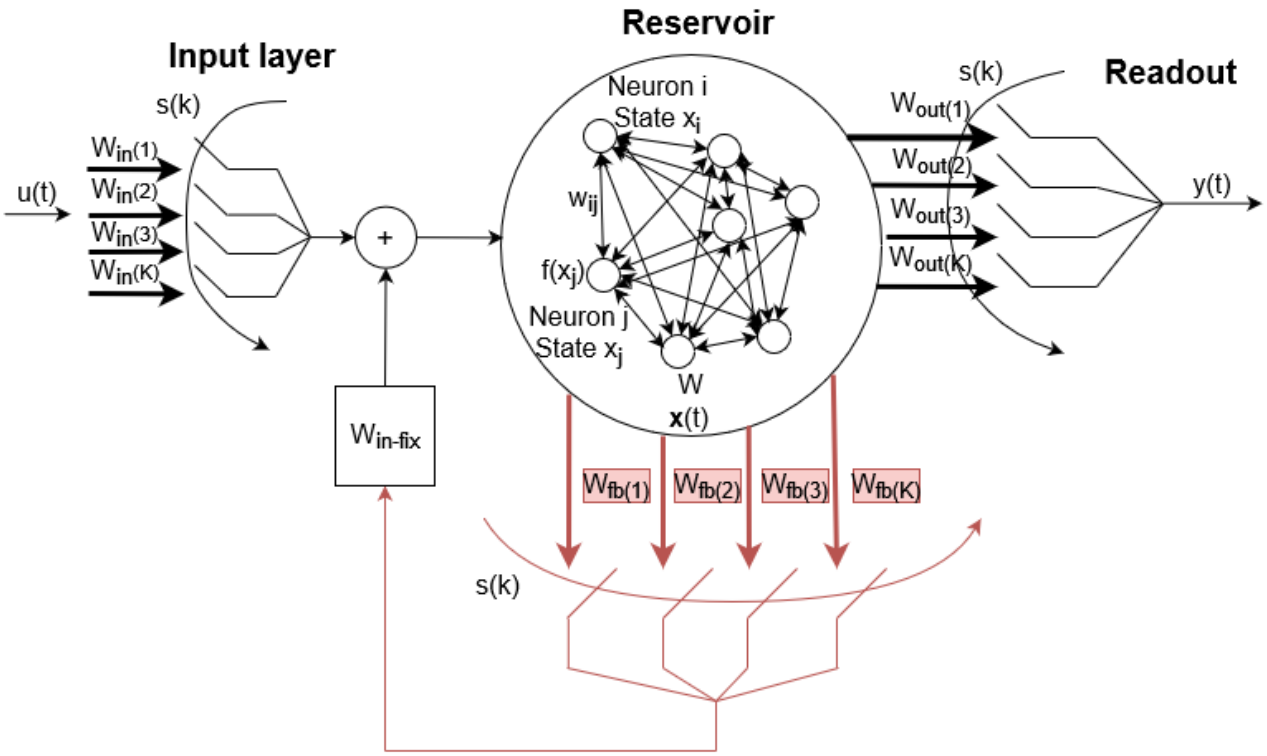

Figure 6.2: **Novel switching ESN architecture and adjustable weights**

6.3 Stability Analysis

This section analyses the stability of the overall switching architecture for linear and non-linear reservoirs. In particular, the following section focuses on the linear case.

6.3.1 Linear reservoirs

Consider the linear discrete-time switching ESN architecture:

$$\begin{aligned} \mathbf{x}(k+1) &= (W - W_{in\,fix} W_{fb_{\sigma(k)}})\mathbf{x}(k) + W_{in_{\sigma(k)}}\mathbf{u}(k) \\ y(k) &= H_{\sigma(k)}(\mathbf{x}(k), W_{out\,\sigma(t)}) \end{aligned} \tag{6.7}$$

with the same notations as in (6.3) and (6.4).

To facilitate the introduction of the stability guarantees, let us recall key theoretical aspects from [132]:

Let $\mathscr{M} = \{M_1, ..., M_K\} \in \mathbb{R}^{n\times n}$ be a set of essentially nonnegative and Schur stable matrices. Consider the map $\pi : \{1, ..., n\} \to \{1, ..., K\}$ with $P = [\pi(1), ., , , .\pi(n)]$ the corresponding n-tuple

and Π *the set of all n-tuples. Construct all the row and column representatives:*

$$\begin{aligned}(M)_P^{\text{row}} &= \begin{bmatrix} \text{row}_1(M_{\pi(1)}) \\ \vdots \\ \text{row}_n(M_{\pi(n)}) \end{bmatrix} \in \mathbb{R}^{n\times n} \\ (M)_P^{\text{col}} &= [col_1(M_{\pi(1)}) \ \ldots col_n(M_{\pi(n)})] \in \mathbb{R}^{n\times n}\end{aligned} \tag{6.8}$$

Subsequently, each row and column representative have an eigenvalue that satisfies:

$$\begin{aligned}\text{Re}\{\lambda_i((M)_P^{\text{row}})\} &\leq \lambda_{PF}((M)_P^{\text{row}}),\ i \in \{1,\ldots,n\} \\ \text{Re}\{\lambda_i((M)_P^{\text{col}})\} &\leq \lambda_{PF}((M)_P^{\text{col}}),\ i \in \{1,\ldots,n\}\end{aligned} \tag{6.9}$$

Define the dominant PF eigenvalues of the row and column representatives:

$$\begin{aligned}\lambda_{\text{repr}}^{\text{row}}(\mathscr{M}) &= \max_{P\in\Pi} \lambda_{PF}((M)_P^{\text{row}}) \\ \lambda_{\text{repr}}^{\text{col}}(\mathscr{M}) &= \max_{P\in\Pi} \lambda_{PF}((M)_P^{\text{col}})\end{aligned} \tag{6.10}$$

If the dominant representative eigenvalues satisfy:

$$\lambda_{\text{repr}}^{\text{row}}(\mathscr{M}) \cdot \lambda_{\text{repr}}^{\text{col}}(\mathscr{M}) < 1 \tag{6.11}$$

then the switching system represented by the set of system matrices $\mathscr{M}$ *is asymptotically globally stable.*

Since the reservoir matrix is randomly initialized, through the aFORCE training algorithm, we can construct closed-loop essentially nonnegative Schur stable matrices with respect to each target system to verify equation (6.11) from [132] as follows:

- **Step 1:** Extend ESN states with the input layer $W_{in_{\sigma(k)}}$ and state-feedback gain $W_{fb_{\sigma(k)}}$ for all K subsystems.
- **Step 2:** Identify the input layer $W_{in_{\sigma(k)}}$ and state-feedback gain $W_{fb_{\sigma(k)}}$ for each subsystem, using the EKF.
- **Step 3:** Construct an essentially nonnegative matrix $W_{nonneg_{\sigma(k)}}$ having the eigenvalues of $W - W_{in_{fix}} W_{fb_{\sigma(k)}}$.
- **Step 4:** Compute the new $W_{in_{fix_{\sigma(k)}}} = (W - W_{nonneg_{\sigma(k)}}) \frac{W_{fb_{\sigma(k)}}^T}{||W_{fb_{\sigma(k)}}||^2}$ to obtain an essentially nonnegative closed-loop system matrix.

In the end, we obtain a set of switching nonnegative Schur stable matrices which can then be used to verify the asymptotic global stability of (6.7) through condition (6.11).

For the aFORCE algorithm, these four steps replace Step 1 and 2 presented in Section 4.4.1.1.

6.3.2 Non-linear differentiable reservoirs

Consider the discrete-time non-linear switching ESN architecture:

$$\begin{aligned} \mathbf{x}(k+1) &= f\left((W - W_{in\,fix}W_{fb_{\sigma(k)}})\mathbf{x}(k) + W_{in_{\sigma(k)}}\mathbf{u}(k)\right) \\ y(k) &= H_{\sigma(k)}(\mathbf{x}(k), W_{out\,\sigma(t)}) \end{aligned} \tag{6.12}$$

with $f : \mathbb{R}^n \to \mathbb{R}^n, f(\mathbf{x}) = tanh(\mathbf{x})$, the same definitions as in (6.3) and (6.4), $\mathbf{x}^* = 0$ the equilibrium point for $\mathbf{u} = 0$ and $\mathbb{B}$ the vecinity of $\mathbf{x}^*$.

Consider the following candidate Lyapunov function:

$$V(\mathbf{x}) = 1 - tanh^2\left((W - W_{in\,fix}W_{fb_{\sigma(k)}})\mathbf{x}(k) + W_{in_{\sigma(k)}}\mathbf{u}(k)\right) \tag{6.13}$$

It is obvious that because $tanh : \mathbb{R} \to [-1, 1]$:

$$V(\mathbf{x}) \geqslant 0, \forall \mathbf{x} \in \mathbb{R} \tag{6.14}$$

Now, consider the derivative:

$$\begin{aligned} &\tfrac{dV}{d\mathbf{x}} = -2tanh^2\left((W - W_{in\,fix}W_{fb_{\sigma(k)}})\mathbf{x}(k) + W_{in_{\sigma(k)}}\mathbf{u}(k)\right) \cdot \\ &sech^2\left((W - W_{in\,fix}W_{fb_{\sigma(k)}})\mathbf{x}(k) + W_{in_{\sigma(k)}}\mathbf{u}(k)\right) \cdot \left((W - W_{in\,fix}W_{fb_{\sigma(k)}})\mathbf{x}(k) + W_{in_{\sigma(k)}}\mathbf{u}(k)\right) \end{aligned} \tag{6.15}$$

The function candidate (6.13) represents a Lyapunov function if equation (6.15) satisfies:

$$\frac{dV}{d\mathbf{x}} < 0, \forall x \in \mathbb{B} - \{0\} \tag{6.16}$$

The latter condition holds if and only if the closed-loop system matrices are positive:

$$(W - W_{in\,fix}W_{fb_{\sigma(k)}}) > 0 \tag{6.17}$$

If conditions (6.14) and (6.17) hold, then (6.13) represents a Lyapunov function for (6.4), which guarantees global asymptotic stability for the non-linear switching ESN system.

This means that, in the aFORCE or aFORCE-redux training procedures, the latter constraint can be integrated as follows:

- **Step 1:** Extend ESN states with the input layer $W_{in_{\sigma(k)}}$ and state-feedback gain $W_{fb_{\sigma(k)}}$ for all K subsystems.
- **Step 2:** Identify the input layer $W_{in_{\sigma(k)}}$ and state-feedback gain $W_{fb_{\sigma(k)}}$ for each subsystem, using the EKF.
- **Step 3:** Construct a positive definite matrix $W_{poz_{\sigma(k)}}$ having the eigenvalues of $W - W_{in_{fix}} W_{fb_{\sigma(k)}}$.
- **Step 4:** Compute the new $W_{in_{fix}\sigma(k)} = (W - W_{poz_{\sigma(k)}}) \frac{W_{fb_{\sigma(k)}}^T}{||W_{fb_{\sigma(k)}}||^2}$ to obtain a positive definite closed-loop system matrix.

For the aFORCE algorithm, these four steps replace Step 1 and 2 presented in Section 4.4.1.1.

In the following section, the numerical experiment is presented to benchmark the new switching architecture against state-of-the-art control-based alternatives.

6.4 Numerical study - Switching ESN approximation of fruitfly photoreceptor on different light intensity levels

This section presents the numerical study using real data from a fruitfly photoreceptor [129] to benchmark the proposed switching ESN architecture.

For this experiment, the input layer is coupled with delays and all the readouts are trained using the original aFORCE algorithm presented in Chapter 4 and consist of non-linear 3rd order polynomials.

Compared to the original condition in the form of Proposition 3.5 from [131], it is more advantageous to use the more relaxed condition presented in equation (6.11) from [132] as it covers a larger class of switching systems.

The data used in this numerical example has been recorded in vivo from the photoreceptor of a fruitfly in the presence of different intensity light stimulus [129]. The light stimuli have been sampled at 2 kHz that have been low-pass filtered by elliptic filters with a 500 Hz cut-off frequency. The authors in [129] present four different intensities that were concatenated, in order to resemble light fluctuations that are present in a flys natural habitat. The light pattern has been selected as a naturalistic time-series of intensities specific to the Van Hateren natural stimuli collection. In Figure 6.3, the input and output response of the fruitfly photoreceptor is presented.

The original signal that has been recorded experimentally from the fruitflies photoreceptor consists of 2 second representative naturalistic time-series of intensities signal sampled initially at a frequency of 500 Hz. The data has then been preprocessed for analysis and experimental purposes to 400 Hz, has been concatenated eight times and the first 200 points representing the transitory regime have been eliminated, as suggested in [129] for single level characterization to ensure that the photoreceptor response is stationary. The eight newly created periods are split as follows: the first six periods are considered for training and model cross-validation. Each period has been sub sectioned as follows: from each period, the first 600 points are used for training purposes and are concatenated into a 3600-point training signal. The last 200 points from each period are concatenated into an 1200-point signal that is used to cross-validate and select the best model that is obtained from the OFR algorithm. The last two periods are used to test the final model.

The next section presents the results using linear switching state-feedback ESNs trained using the aFORCE algorithm.

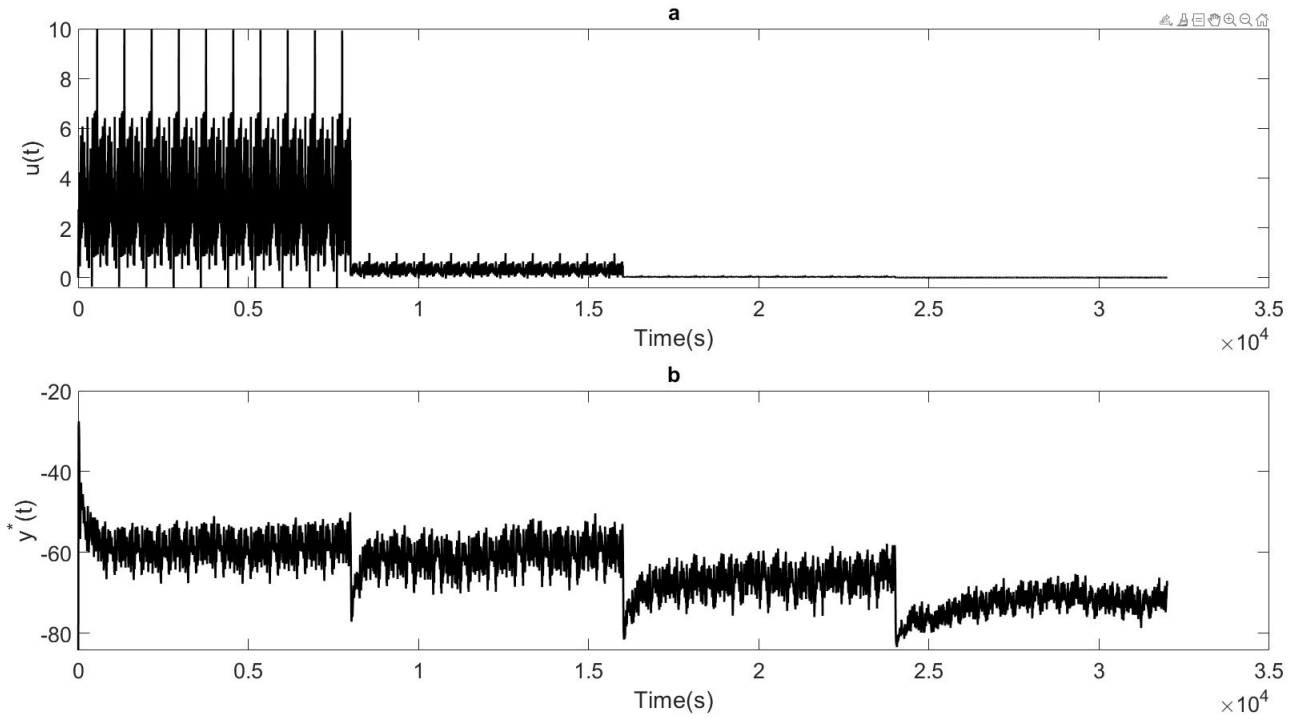

Figure 6.3: **Experimental data recordings of** a) naturalistic stimuli and b) in vivo Fruitfly photoreceptor responses used to fit the ESN model

6.4.1 Linear reservoir

This section presents the numerical results for linear switching state-feedback ESNs trained using the aFORCE algorithm.

Table 6.1 shows the resulted NMSE using the original aFORCE algorithm for a linear SESN with respect to the 4 indicated levels of light intensity fed to the photoreceptor of the fruitfly. Note that the NMSE reported in [129] for the NARMAX model is 0.85. In this context, it is worth noting that the proposed architecture outperforms the NARMAX model proposed in [129] for all combinations of reservoir dimensionality and input delays.

	Linear SESN aFORCE NMSE						
Dimensionality/Delay	**1**	**2**	**3**	**4**	**5**	**6**	**7**
11	0.55	0.38	0.34	0.27	0.21	0.21	0.23
12	0.24	0.32	0.27	0.29	0.26	0.2	0.2
13	0.49	0.28	0.26	0.26	0.21	0.2	0.22
14	0.28	0.36	0.28	0.26	0.22	0.17	0.22

Table 6.1: **Linear SESN NMSE** with regards to different reservoir dimensionalities, different number of input delays and 1 output lag

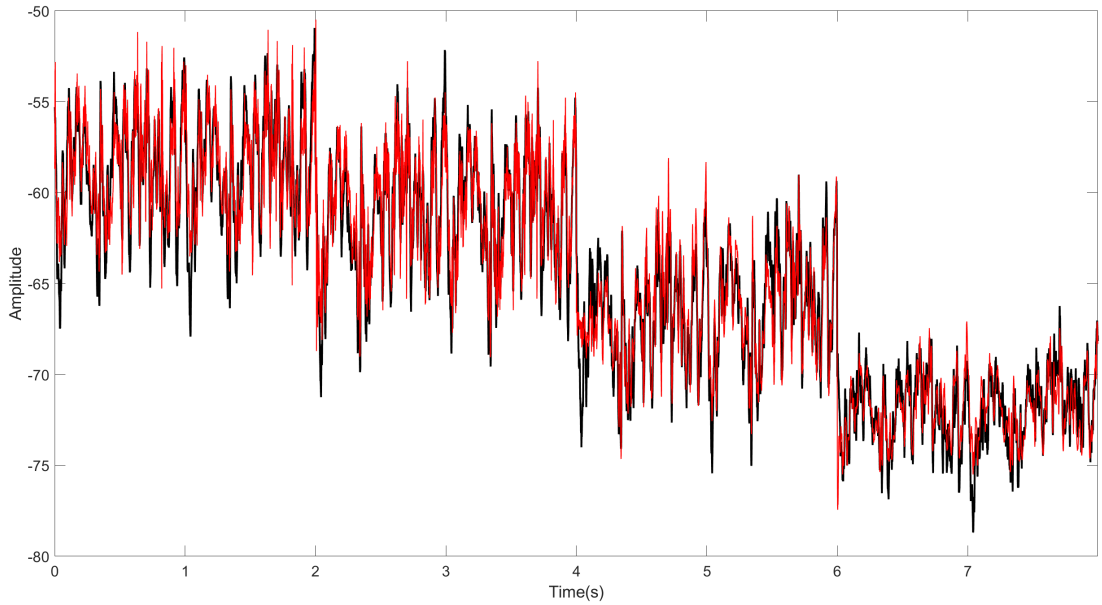

Figure 6.4: **ESN response to real-life data recorded from the photoreceptor of a Fruit Fly**: ESN model predicted output (red) superimposed on experimental photoreceptor responses (black) for a linear reservoir having $N = 14$ neurons, maximum input lags $l = 5$ and 1 output lag with $NMSE = 0.19$

6.4.2 Non-linear invertible reservoir

This section presents the numerical results for linear switching state-feedback ESNs trained using the aFORCE algorithm.

Table 6.2 shows the resulted NMSE using the original aFORCE algorithm for a non-linear SESN with respect to the 4 indicated levels of light intensity fed to the photoreceptor of the fruitfly. Note that the NMSE reported in [129] for the NARMAX model is 0.85. In this context, it is worth noting that the proposed architecture outperforms the NARMAX model proposed in [129] for all combinations of reservoir dimensionality and input delays.

	Non-linear SESN aFORCE NMSE						
Dimensionality/Delay	**1**	**2**	**3**	**4**	**5**	**6**	**7**
11	0.33	0.34	0.18	0.23	0.41	0.14	0.49
12	0.24	0.32	0.27	0.21	0.21	0.18	0.2
13	0.25	0.23	0.38	0.23	0.24	0.32	0.15
14	0.28	0.23	0.12	0.15	0.31	0.17	0.22

Table 6.2: **Non-linear SESN NMSE** with regards to different reservoir dimensionalities, different number of input delays and 1 output lag

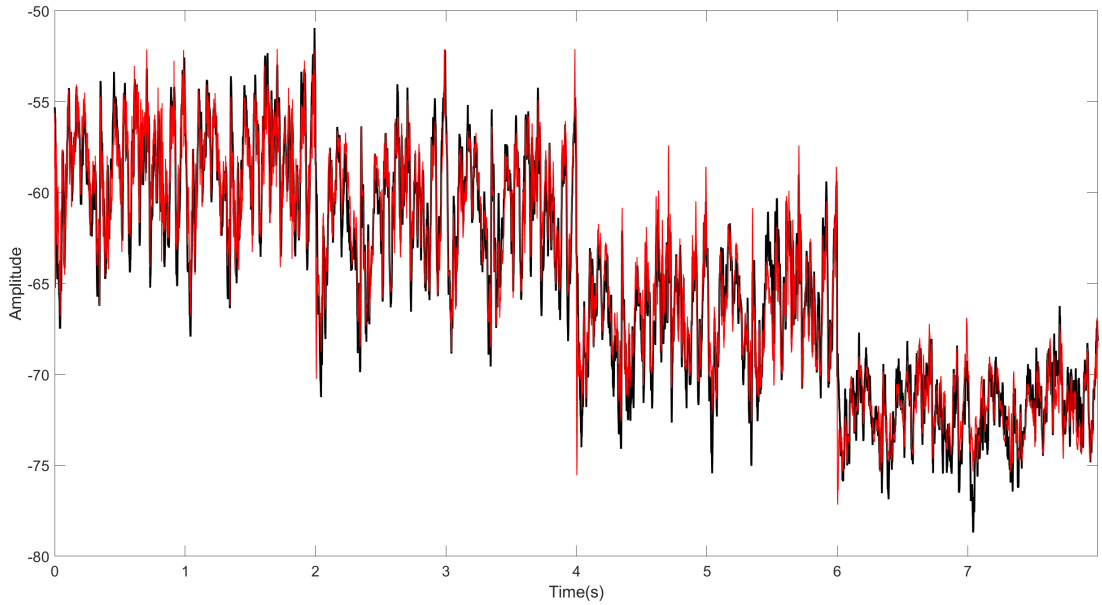

Figure 6.5: **ESN response to real-life data recorded from the photoreceptor of a Fruit Fly**: ESN model predicted output (red) superimposed on experimental photoreceptor responses (black) for a non-linear reservoir having $N = 13$ neurons, maximum input lags $l = 4$ and 1 output lag with $NMSE = 0.14$

Comparing Tables 6.2 and 6.1 shows that the non-linear reservoir SESN provides better approximation accuracy over the linear SESN. However, the main advantage of the linear SESN is in its monotonic behavior with respect to the NMSE when increasing the number of neurons and delays. In contrast, the non-linear SESN exhibits a non-monotonic behavior for the same scenario. For instance, in Table 6.2 it can be seen that for $N = 13$, increasing the delay from 2 to 3 results in higher a higher valued NMSE.

The following section presents the concluding remarks of the chapter.

6.5 Conclusions

This chapter proposes a new type of ESN architecture that uses switching input layers, state-feedback gains and readout modules for processing multiple tasks. Training the architecture is done via the aFORCE algorithm proposed in Chapter 4, which uses an alternative training strategy for computing the input layer and state-feedback gain via an EKF estimation method [20],[21],[22] and finds the optimal parameters of a polynomial readout module via an OFR algorithm. The selection of the state-feedback gain and input layer allows the closed-loop structure to better approximate the dynamics of target system, which enhances the outputs fitting performance in the second stage of the aFORCE algorithm.

Section 6.2 presents an in-depth analysis of the overall stability of the switching system, for both linear and non-linear reservoirs. It has been shown that the standard non-linear reservoir structure (e.g., the hyperbolic tangent activation function) guarantees asymptotical stability on the entire definition domain of the *tanh* function via Lyapunov function, when the closed-loop subsystem matrices are positive definite. In the case of linear reservoirs when Lyapunov functions cannot be found, stability can still be verified through other means [132]. In the first instance, applying an EKF [21],[22] to compute the input layer and state-feedback gain allows finding appropriate eigenvalues for fitting the dynamics of the unknown target tasks. Then, computing the decoupled state-feedback gain ensures essentially non-negative closed-loop system matrices [134],[135]. This allows for a convenient stability verification, namely, on the dominant row and column eigenvalues earlier defined in equation (6.11) from [132].

Based on the theoretical analysis presented in Chapter 4, the SESN architecture outperforms standard ESNs coupled with parallel readouts. The main drawback however, is that the computations of each task have to be carried out serially due to the switching nature of the input layer and state-feedback structures that modify the dynamics of the reservoir. Although this approach leads to increased running times, SESN provides higher output approximation accuracy.

The numerical study used real world data extracted from the photoreceptor of a fruitfly in the presence of different intensity light stimulus [129]. In all considered scenarios, the numerical results show that the proposed SESN architecture trained using the aFORCE method performs better than the NARMAX approach proposed in [129]. The simulations also highlight that the non-linear reservoir SESN performs better than the linear SESN. However, the performance of the linear SESN is more consistent across reservoir dimensionality and input delay regimes.

The next chapter draws the final conclusions of the thesis and identifies future work directions based on the current findings.

Chapter 7

Conclusions and future work

The National Academy of Engineering in the US published a list in 2008 regarding the grand challenges for engineering in the 21st century. From the 14 challenges that have been identified in their report, one is of particular interest for the subject debated in this thesis and it consists in the "reverse engineering" of the brain. However, in order to understand how the brain works, an in-depth analysis of current theoretical frameworks as well as the development of new theoretical results and architectures are required for inferring the general principles and underlying particular mechanisms of the brains functionality.

To this end, this thesis has proposed a new in depth analysis on the theoretical framework that addresses the ability of current Reservoir Computing architectures to represent Universal Approximators, which revealed some limitations regarding their approximation performance. Training only a structure that is used to deal with two separate problems, namely, output signal matching and internal dynamical fitting with respect to a target task, does not guarantee that these competing goals can be achieved. Ultimately, this leads to a compromise solution for the readout module, which introduces some lower boundaries to the approximation capability of the reservoir.

This new discovery led to the development of two new training algorithms for the underdeveloped state-feedback Echo State Network architectures, namely, the aFORCE and the aFORCE-redux, that efficiently train the state-feedback and input layer. An in depth theoretical analysis revealed that separating the two aforementioned problems of internal dynamical fitting

and output signal approximation via two sepparate trainable structures, namely, a state-feedback gain and a readout module, could, in theory, offer perfect approximation results. These facts are backed up by the numerical simulations that show the improved accuracy of state-feedback Echo State Networks trained using either of the proposed training algorithms over the state-of-the-art output-feedback Echo State Network models and their afferent training strategies, such as the FORCE. Moreover, the ability to select the most relevant neurons that dictate the behavior of the reservoir with respect to the dynamics of the target task, in the case of the aFORCE-redux algorithm, as well as the components of the polynomial readout module increases the noise robustness of the overall structure and lowers the risk of overfitting the trainable parameters in certain scenarios. For some cases, lower dimension reservoirs with less trainable parameters trained using the aFORCE-redux perform better compared to higher dimension reservoirs that have fully trained parameters trained using the original aFORCE. However, in some examples, it is still beneficial to train all the parameters using the aFORCE method, as the approximation results are improved when compared to the more time and space efficient aFORCE-redux method.

Analyzing the proposed training algorithms and architectures reveals certain limitations:

1. Using the EKF estimation for training the state-feedback gain and the input layer of ESNs via aFORCE and aFORCE-redux does not guarantee optimal solutions in the case of approximating non-linear tasks. Because the EKF approximates state transitions and measurements using linear Taylor expansions, the approximation accuracy is dependent on the degree of non-linearity of the functions that are approximated.

2. Although SESN offers better approximation results when compared to the traditional parallel ESN, due to the nature of serially computing each task, SESN is constrained by the complexity of the tasks, their number, the frequency of switching and by the computational power of the machine it operates on. A large number of complex target tasks with a high-frequency of switching do not recommend using SESN when the main goal is real-time computing, as the required computational power is higher to that of less accurate parallel architectures.

All computations have been made using a i7-10700K chip-set coupled with 32GB of DDR4 3600Mhz RAM.

In order to maximize the training capacities of current algorithms that mimic the brain computation, the huge amount of neuroscience data that is available can be exploited for developing new types of architectures and training strategies that eliminate the downsides of current Reservoir Computing architectures and, even benefit from certain advantages observed in the biological brain.

A better understanding of how the brain works and how it is internally organised will not only enable the treatment of damaged brains, the enhancement of brains via wearable devices, or the

improvement of machine learning techniques, but will also enable the development of microchips that could mimic the neural architecture and implement similar training strategies.

The results presented in this thesis could be further developed is a number of ways.

1. Similar to the Orthogonal Forward Selection methods that have been used in Chapters 4 and 5, other efficient selection methods could be applied, which could improve the proposed training methods for certain tasks. It would be beneficial to explore other mathematical algorithms related to parameter selection and design new training methodologies specifically for certain task-related constraints.

2. The training algorithms introduced in Chapters 4 and 5 are developed exclusively for Echo State Networks with discrete linear and non-linear neurons. It would be extremely useful to extend the framework to other types of networks and architectures and to the more complex neuron models that operate in the spiking domain.

3. The new switching Echo State Network architecture introduced in Chapter 6 is stable only when the closed-loop reservoir meets a set of constraints, related to the type of the closed-loop system matrix. It would be interesting to further analyse and develop a set of more relaxed conditions. Moreover, an external switching sequence could be designed to eliminate the constraints entirely.

4. The literature would benefit from advancements regarding the development of new architectures by continuing the work developed in Chapter 6. The perspective of switching can be expanded to network architectures that use other types of neural activation functions, such as Liquid State Machines or to entirely other types of architectures, such as Deep Neural Networks.

Bibliography

1. Vaishya, R., Javaid, M., Khan, I. H. & Haleem, A. Artificial Intelligence (AI) applications for COVID-19 pandemic. *Diabetes and Metabolic Syndrome: Clinical Research and Reviews* **14,** 337–339 (2020).
2. Beck, J., Stern, M. & Haugsjaa, E. Applications of AI in Educatio. *XRDS: Crossroads, The ACM Magazine for Students* **3,** 11–15 (1996).
3. AL-Dosari, K., Fetais, N. & Kucukvar, M. Artificial Intelligence and Cyber Defense System for Banking Industry: A Qualitative Study of AI Applications and Challenges. *Cybernetics and Systems,* 1–29 (2022).
4. Murphy, K. P. *Machine Learning A probabilistic perspective* (The MIT Press, London, 2012).
5. Haykin, S. *Neural Networks and Learning Machines* (Pearson Prentice Hall, New York, 1999).
6. Haykin, S. *Adaptive Filter Theory* (Prentice Hall, 2002).
7. S. Chen, S. B. Neural networks for nonlinear dynamic system modelling and identification. *International Journal of Control* **56,** 319–346 (1992).
8. Maas, W., Natschläger, T. & Markram, H. The "Liquid Computer": A Novel Strategy for Real-Time Computing on Time Series. *Special Issue on Foundations of Information Processing of TELEMATIK* **8,** 39–43 (2002).
9. Jaeger, H. & Haas, H. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science,* 78–80 (2004).
10. Huang, G. *et al.* Trends in extreme learning machines: A review. *Neural Networks* **61,** 32–48 (2015).
11. McDonald, N. Reservoir computing and extreme learning machines using pairs of cellular automata rules. *2017 International Joint Conference on Neural Networks (IJCNN),* 2429–2436 (2017).

12. Sussillo, D. & Abott, L. Generating Coherent Patterns of Activity from Chaotic Neural Networks. *Neuron* **63,** 544–557 (2009).

13. Maas, W. *Liquid State Machines: Motivation, Theory and Applications* 275–296 (Imperial College Press, London, 2011).

14. Maass, W., Joshi, P. & Sontag, E. Computational Aspects of Feedback in Neural Circuits. *PLoS Computational Biology* **3,** 15–34 (2007).

15. Maas, W., Natschläger, T. & Markram, H. Real-Time Computing Without Stable States: A New framework for Neural Computation based on Perturbations. *Neural Computation* **14,** 2531–2560 (2002).

16. Maas, W. To Spike or Not to Spike: That Is The Question. *Proceedings of the IEEE* **103,** 2219–2224 (2015).

17. Maas, W. Noise as a Resource for Computation and Learining in Networks of Spiking Neurons. *Proceedings of the IEEE* **102,** 860–880 (2014).

18. Lukosevicius, M. & Jaeger, H. Reservoir computing approaches to recurrent neural network training. *Computer Science Review* **3,** 127–149 (2009).

19. Gauthier, D. *et al.* Next generation reservoir computing. *Nature Communications* **12,** 5564 (2021).

20. Sorenson, H. W. *Kalman Filtering: Theory and Application* (Piscataway, NJ: IEEE, 1982).

21. Ribeiro, M. Kalman and extended kalman filters: Concept, derivation and properties. *Institute for Systems and Robotics)* **43** (2004).

22. Irshad, A. *et al.* Extended state space recursive least squares. *Digital Signal Processing* **49,** 95–103 (2015).

23. Chen, S., Billings, S. A. & Luo, W. Orthogonal least squares methods and their application to non-linear system identification. *International Journal of Control* **50,** 1873–1896 (1989).

24. Florescu, D. *Reconstruction, Identification and Implementation Methods for Spiking Neural Networks* (Springer, 2017).

25. Schuster, M. & Paliwal, K. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing* **45,** 2673–2681 (1997).

26. Kim, J. *et al.* Teaching recurrent neural networks to infer global temporal structure from local examples. *Nature Machine Intelligence* **3,** 316–323 (2021).

27. LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. *Nature* **521,** 436–444 (2015).

28. Sussillo, D. Neural circuits as computational dynamical systems. *Current Opinion in Neurobiology* **25,** 156–163 (2015).

29. Dupond, S. A thorough review on the current advance of neural network structures. *Annual Reviews in Control* **14,** 200–230 (2019).

30. Abiodun, O. *et al.* State-of-the-art in artificial neural network applications: A survey. *Heliyon* **4** (2018).

31. Maas, W. & Sonntag, E. D. Analog neural nets with gaussian or other common noise distribution cannot recognize arbitrary regular languages. *Neural Computation* **11,** 771–782 (1999).

32. Florescu, D. & Coca, D. Learning with Precise Spike Times: A New Decoding Algorithm for Liquid State Machines. *Neural Computation,* 1825–1852 (2019).

33. Jin, L. & Gupta, M. Stable dynamic backpropagation learning in recurrent neural networks. *IEEE Transactions on Neural Networks* **10,** 1321–1334 (1999).

34. Atiya, A. & Parlos, A. New results on recurrent network training: unifying the algorithms and accelerating convergence. *IEEE Transactions on Neural Networks* **11,** 697–709 (2000).

35. Wan, E. & Nelson, A. *Dual extended Kalman filter methods* (Wiley, New York, 2001).

36. Merikitani, D. & Nikolaev, N. Recurrent Expectation Maximization Neural Modeling. *International Conference on Computational Intelligence for Modelling Control and Automation,* 674–679 (2008).

37. Williams, R. & Zipser, D. A learning algorithm for continually running fully recurrent neural networks. *Neural Comput.* **1,** 270–280 (1989).

38. Robinson, A. & Fallside, F. The utility driven dynamic error propa- gation network. *Cambridge Univ. Eng. Dept., Tech. Rep.* (1987).

39. Rumelhart, D., Hinton, G. & Williams, R. Learning representations by back-propagating errors. *Nature* **323,** 533–536 (1986).

40. Werbos, P. Backpropagation through time: What it does and how to do it. *Proceedings of IEEE Transactions on Neural Networks* **11,** 279–297 (1990).

41. Pearlmutter, B. Learning state-space trajectories in recurrent neural net- works. *Neural Comput.* **1,** 263–269 (1989).

42. Toomarian, N. & Barhen, J. Adjoint-functions and temporal learning algorithms in neural networks. *Advances in Neural Information Processing Systems,* 113–120 (1991).

43. G.-Z. Sun, H.-H. C. & Lee, Y.-C. Green's function method for fast on-line learning algorithm of recurrent neural networks. *Advances Neural Inform. Processing Syst,* 333–340 (1992).

44. Schmidhuber, J. A fixed storage O(N^3) time complexity learning algorithm for fully recurrent continually running networks. *Neural Comput.* **4,** 243–248 (1992).

45. Williams, R. *Complexity of exact gradient computation algorithms for recurrent neural networks* (1992).

46. Sheng, C., Zhao, J., Liu, Y. & Wang, W. Prediction for noisy nonlinear time series by echo state network based on dual estimation. *Neurocomputing* **82,** 186–195 (2012).

47. Peng, X., Dong, H. & Zhang, B. Echo State Network ship motion modeling prediction based on Kalman filter. *2017 IEEE International Conference on Mechatronics and Automation (ICMA),* 95–100 (2017).

48. Tiňo, P. & Mills, A. J. Learning beyond finite memory in recurrent networks of spiking neurons. *Neural computation* **18,** 591–613 (2006).

49. Diehl, P. U., Zarrella, G., Cassidy, A., Pedroni, B. U. & Neftci, E. Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware. *2016 IEEE International Conference on Rebooting Computing (ICRC),* 1–8 (2016).

50. Huh, D. & Sejnowski, T. J. Gradient descent for spiking neural networks. *Advances in neural information processing systems* **31** (2018).

51. Pfister, J.-P., Toyoizumi, T., Barber, D. & Gerstner, W. Optimal spike-timing-dependent plasticity for precise action potential firing in supervised learning. *Neural computation* **18,** 1318–1348 (2006).

52. Brea, J., Senn, W. & Pfister, J.-P. Sequence learning with hidden units in spiking neural networks. *Advances in neural information processing systems* **24** (2011).

53. Brea, J., Senn, W. & Pfister, J.-P. Matching recall and storage in sequence learning with spiking neural networks. *Journal of neuroscience* **33,** 9565–9575 (2013).

54. Memmesheimer, R., Rubin, R., Olveczky, B. & Sompolinsky, H. To spike, or when to spike? *Neuron* **82,** 925–938 (2014).

55. Gilra, A. & Gerstner, W. Predicting non-linear dynamics by stable local learning in a recurrent spiking neural network. *Neuroscience eLIFE* (2017).

56. Luo, Y. *et al.* Improving the stability for spiking neural networks using anti-noise learning rule, 29–37 (2018).

57. Savage, J. *Models ofcomputation: Exploring the power ofcomputing* (Reading, MA: Addison-Wesley, 1998).

58. Maas, W. Networks of Spiking Neurons: The Third Generation of Neural Networks Model. *Neural Networks* **10,** 1659–1671 (1997).

59. Legendre, A. *Nouvelles méthodes pour la détermination des orbites des comètes* (F. Didot, 1805).

60. Kollias, S. & Anastassiou, D. An adaptive least squares algorithm for the efficient training of artificial neural networks. *IEEE Transactions on Circuits and Systems* **36,** 1092–1101 (1989).

61. Martens, J. & Grosse, R. Optimizing Neural Networks with Kronecker-factored Approximate Curvature. *Proceedings of the 32nd International Conference on Machine Learning* **37,** 2408–2417 (2015).

62. Schmidhuber, J., Gagliolo, M., Wierstra, D. & Gomez, F. Evolino for Recurrent Support Vector Machines. *European Symposium on Artificial Neural Networks,* 593–598 (2005).

63. Shi, Z. & Han, M. Support vector echo-state machine for chaotic time-series prediction. *IEEE Transactions on Neural Networks* **18,** 359–372 (2007).

64. Broomhead, D. & Lowe, D. Multivariable Functional Interpolation and Adaptive Networks. *Complex Systems* **2,** 321–355 (1988).

65. Wiskott, L. & Sejnowski, T. Slow feature analysis: Unsupervised learning of invariances. *Nueral Computation* **14,** 715–770 (2002).

66. Basios, N. & Bors, A. Variational learning for Gaussian mixture models. *IEEE Transactions on Systems Manufacturing and Cybernetics, Part B (Cybernetics)* **36,** 849–862 (2006).

67. Gonon, L. & Ortega, J. Reservoir computing universality with stochastic inputs. *IEEE Transactions: Neural Networks and Learning Systems* **31,** 100–112 (2020).

68. Hart, A., Hook, J. & Dawes, J. Echo state networks trained by Tikhonov least squares are L2 approximators of ergodic dynamical systems. *Physica D: Nonlinear Phenoma* **421** (2021).

69. Bollt, E. On explaining the surprising success of reservoir computing forecaster of chaos? The universal machine learning dynamical system with contrast to VAR and DMD. *Chaos* **31** (2021).

70. Boyd, S. & Chua, L. O. Fading Memory and the Problem of Approximating Nonlinear Operators with Volterra Series. *IEEE Transactions On Circuits and Systems* **32,** 1150–1161 (1985).

71. Kolmogorov, A. The representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. *Doklady Akademii Nauk SSSR* **114,** 953–956 (1957).

72. Cybenko, G. Approximation by Superpositions of a Sigmoidal Function. *Math. Control Signals Systems* **2,** 303–314 (1989).

73. K. Hornik M. Stinchcombe, H. W. Multilayer Feedforward Networks are Universal Approximators. *Neural Networks* **2,** 359–366 (1989).

74. Leshno, M. *et al.* Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks* **6,** 5861–867 (1993).

75. Pinkus, A. Approximation theory of the MLP model in neural networks. *Acta Numerica* **8,** 143–195 (1999).

76. Gripenberg, G. Approximation by neural networks with a bounded number of nodes at each level. *Journal of Approximation Theory* **122,** 260–266 (2003).

77. Yarotsky, D. Error bounds for approximations with deep ReLU networks (2017).

78. Zhou, L. *et al.* The Expressive Power of Neural Networks: A View from the Width. *Advances in Neural Information Processing Systems* **30,** 6231–6239 (2017).

79. B. Hanin, M. S. Approximating Continuous Functions by ReLU Nets of Minimal Width (2018).

80. Park, S. *et al.* Minimum Width for Universal Approximation. *International Conference on Learning Representations* (2021).

81. N. Guliyev, V. I. Approximation capability of two hidden layer feedforward neural networks with fixed weights. *Neurocomputing* **316,** 262–269 (2018).

82. Qian, X. & Yoon, B. Effective identification of conserved pathways in biological networks using hidden Markov models. *PLoS ONE* **4,** e8070 (2009).

83. Aarts, E. & Korst, J. *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing* (Wiley, New York, 1989).

84. Carnell, A. & Richardson, D. Linear algebra for time series of spikes. *Proc. of ESANN,* 363–368 (2005).

85. Davidson, S. & Furber, S. Comparison of Artificial and Spiking Neural Networks on Digital Hardware. *Frontiers in Neuroscience* **15** (2021).

86. Lapicque, L. Recherches quantitatives sur l'excitatione electrique des nerfs traitee comme une polarization. *J. Physiol. Pathol.* **9,** 620–635 (1907).

87. Florescu, D. & Coca, D. A Novel Reconstruction Framework for Time-Encoded Signals with Integrate-and-Fire Neurons. *Neural Computation,* 1872–1898 (2015).

88. Naud, R. & Gerstner, W. *The Performance and Limits of Simple Neuron Models: Generalizations of the LIF Model* (Springer, 2012).

89. Teka, W., Marinov, T. & Santamaria, F. Neuronal Spike Timing Adaptation Described with a Fractional Leaky Integrate-and-Fire Model. *PLoS Computational Biology* **10** (2014).

90. Grzyb, B. J. *et al.* Which Model to Use for the Liquid State Machine? *Conference: International Joint Conference on Neural Networks* (2009).

91. Hodgkin, A. & Huxley, A. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology* **117,** 500–544 (1952).

92. Wilson, H. *Spikes, Decisions and Actions* (Oxford University Press, New York, 1999).

93. Izhikevich, E. *Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting* (The MIT Press, Cambridge, Massachusetts, 2010).

94. Wojcik, G. & Kaminski, W. Liquid State Machine Built of Hodgkin- Huxley Neurons and Pattern Recognition. *Neurocomputing* **239,** 245–251 (2004).

95. Wojcik, G. & Kaminski, W. Liquid State Machine and its separation ability as function of electrical parameters of cell. *Neurocomputing* **70,** 2593–2697 (2007).

96. Grzyb, B. *et al.* Facial Expression Recognition based on Liquid State Machines built of Alternative Neuron Models. *Intl. Joint Conf. on Neural Networks* (2009).

97. Morris, C. & Lecar, H. Voltage oscillations in the barnacle giant muscle fiber. *Biophysical Journal,* 193–213 (1981).

98. Zheng, Q. & Wei, G. Poisson–Boltzmann–Nernst–Planck model. *Journal of Chemical Physics* **134** (2011).

99. FitzHugh, R. Impulses and physiological states in theoretical models of nerve membrane. *Biophysical Journal,* 257–278 (1981).

100. Izhikevich, E. Simple Model of Spiking Neurons. *IEEE Transac- tions on Neural Networks* **14,** 1569–1572 (2003).

101. Lazar, A. Information representation with an ensemble of Hodgkin-Huxley Neurons. *Neurocomputing* **70,** 1764–1771 (2007).

102. Bjorck, A. *Numerical methods for least squares problems* (Society for Industrial and Applied Mathematics, 1996).

103. Gerstner, W. *et al.* A neuronal learning rule for sub-millisecond temporal coding. *Nature* **386,** 76–78 (1996).

104. Kempter, R., Gerstner, W. & van Hemmen, J. L. Hebbian learning and spiking neurons. *Phys. Rev. E* **59,** 4498–4514 (1999).

105. Hebb, D. *The Organization of Behavior: A Neuropsychological Theory* (Psychology Press, New York, 2002).

106. Oliveri, A., Rizzo, R. & Chella, A. An Application of Spike-Timing-Dependent Plasticity to Readout Circuit for Liquid State Machine, 1441–1445 (2007).

107. Schrauwen, B., Verstraeten, D. & Campenhout, J. An overview of reservoir computing: Theory, applications and implementations. *Proceedings of the 15th European Symposium on Artificial Neural Networks,* 471–482 (2007).

108. Lukosevicius, M. Echo State Networks with Trained Feedbacks. *Jacobs University Technical Reports* (2007).

109. Ozturk, M. C., Xu, D. & Príncipe, J. Analysis and design of echo state networks. *Neural Computation* **18,** 1364–1375 (2007).

110. Xia, Y. *et al.* An augmented echo state network for nonlinear adaptive filtering of complex noncircular signals. *IEEE Transactions: Neural Networks* **22,** 74–83 (2011).

111. Chatzis, S. & Demiris, Y. Echo state Gaussian process. *IEEE Transactions: Neural Networks* **22,** 1435–1445 (2011).

112. Moller, M. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks* **6,** 525–533 (1993).

113. Anantharaman, R. *et al.* Accelerating Simulation of Stiff Nonlinear Systems using Continuous-Time Echo State Networks. *arXiv preprint arXiv:2010.04004 (2020).*

114. Bozhkov, L., Koprinkova-Hristova, P. & Georgieva, P. Learning to decode human emotions with Echo State Networks. *Neural Networks* **78,** 112–119 (2016).

115. Nicola, W. & Clopath, C. Supervised Learning in Spiking Neural Networks with FORCE Training. *Nature Communications* **8,** 1–15 (2016).

116. Hayes, H. M. *9.4: Recursive Least Squares". Statistical Digital Signal Processing and Modeling* (Wiley, 1996).

117. Naylor, A. W. & Sell, G. R. *Linear Operator Theory in Engineering and Science* (Springer-Verlag, New York, 1982).

118. Pineda, F. Generalization of back-propagation to recurrent neural networks. *Physical Review Letter* **59,** 2229 (1986).

119. Jordan, M. Generic constraints on underspecified target trajectories. *International 1989 Joint Conference on Neural Networks* **1,** 217–225 (1989).

120. Elman, J. Finding Structure in Time. *Cognitive Science* **14,** 179–211 (1990).

121. Kietzmann, T. *et al.* Recurrence is required to capture the representational dynamics of the human visual system. *PNAS* **116,** 21854–21863 (2019).

122. Bengio, Y., Simard, P. & Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks* **5,** 157–166 (1994).

123. Sussillo, D. & Barak, O. Opening the black box: low-dimensional dynamics in high-dimensional recurrent neural networks. *Neural Computation* **25,** 626–649 (2013).

124. Tanaka, G. *et al.* Recent advances in physical reservoir computing: A review. *Neural Networks* **115,** 100–123 (2019).

125. Deng, Y. & Zhang, Y. Collective behavior of a small-world recurrent neural system with scale-free distribution. *IEEE Transactions: Neural Networks* **18,** 1364–1375 (2007).

126. Huang, L. *et al.* Task Learning Promotes Plasticity of Interneuron Connectivity Maps in the Olfactory Bulb. *Journal of Neuroscience* **36,** 8856–8871 (2016).

127. Terejanu, G. Extended Kalman filter tutorial. *University at Buffalo* (2008).

128. Shine, J. *et al.* Human cognition involves the dynamic integration of neural activity and neuromodulatory systems. *Nature Neuroscience* **22,** 289–296 (2019).

129. Uwe, F. *et al.* Fly Photoreceptors Encode Phase Congruency. *PLoS One* **11** (2016).

130. Aubin, B. *et al.* The committee machine: Computational to statistical gaps in learning a two-layers neural network. *Advances in Neural Information Processing Systems* **31** (2018).

131. Blanchini, F. & Miani, S. *Set-Theoretic Methods in Control* (Birkhauser, Boston, 2008).

132. Lupascu, C., Nechita, S. & Pastravanu, O. Dual switched positive systems – a less conservative condition for diagonal quadratic stability. *International Journal of Systems Science* **50,** 2529–2538 (2019).

133. Global input-to-state stability and stabilization of discrete-time piecewise affine systems. *Nonlinear Analysis: Hybrid Systems* **2,** 721–734 (2008).

134. Soto, R. Existence and construction of nonnegative matrices with prescribed spectrum. *Linear Algebra and its Applications* **369,** 169–185 (2001).

135. Arietta, L., Millano, A. & Soto, R. On spectra realizable and diagonalizably realizable. *Linear Algebra and its Applications* **612,** 273–288 (2021).