Università di Pisa

Dipartimento di Informatica
Dottorato di Ricerca in Informatica

Ph.D. Thesis

Deep Reservoir Computing

A Novel Class of Deep Recurrent Neural Networks

Luca Pedrelli

Supervisors

Prof. Alessio Micheli

Ph.D. Claudio Gallicchio

Contents

Abstract

In this thesis we propose a novel class of deep Recurrent Neural Networks (RNNs) explicitly extending the Reservoir Computing framework to the Deep Learning paradigm. Thereby, we introduce the Deep Echo State Network (DeepESN) model characterized by a hierarchy of randomized recurrent layers.
The introduction of randomized deep RNNs has provided tools to analyze deep recurrent models separately from learning algorithms aspects.
The analysis and the experimental assessments conducted on DeepESNs highlighted that layering in deep RNNs is intrinsically able to develop hierarchical, distributed temporal features.
We evaluated our approach on controlled scenarios and challenging real-world tasks. Overall, DeepESN models allowed us to design extremely efficient deep RNNs that obtained performance competing with state-of-the-art approaches.

Acknowledgements

I thank my supervisors Alessio Micheli and Claudio Gallicchio for their precious teachings. They guided and encouraging me to tackle scientific problems with great enthusiasm. Their contribution to my personal growth in the field of the scientific research is invaluable. I hope to repay their help with my contributions in future works on challenging problems. I thank Igor Farkaš and Peter Tiňo for providing reviews with relevant comments and observations which helped me to improve the quality of the thesis. I thank Pierpaolo Degano who has been the coordinator of the graduate study program for many years. I thank the actual coordinator of the graduate study program, Paolo Ferragina and the members of my thesis commettee, Salvatore Ruggieri and Stefano Chessa. Finally, I thank all the co-authors of the works which contribuited to this thesis.

Ringraziamenti

Ringrazio tutta la mia famiglia che mi ha sempre aiutato. Ringrazio tutti i miei amici che mi hanno accompagnato nel viaggio. Ringrazio il Dipartimento di Matematica dell'Università di Parma che ha formato le mie conoscenze scientifiche e il Dipartimento di Informatica dell'Università di Pisa per avermi fatto entrare nel mondo della ricerca scientifica. Infine ringrazio la città di Pisa che ritengo essere un ottimo ambiente culturare ed artistico dove coltivare interessi preziosi ed acquisire gradi capacità.

Introduction

1.1 Motivations

In the last years, the design of novel architectures of *neural networks* aroused a great interest in the *machine learning* research community. The aim of designing neural network architectures is to provide rich and proper inner representations of the input information in order to deal with tasks for which the information is characterized by a compositional nature or by complex relationships between entities. Since many real-world tasks are characterized by such complexity, the aim of a proper design is relevant to improve the performance of the models.

In this thesis, we are interested in sequential domain applications. Such tasks consist in processing data composed by sets of sequences. A sequence is a collection of entities and an entity is represented by a feature vector. Moreover, the entities that belong to a sequence have a relationship of serial order. In particular, if the relationships represent temporal dependencies then the sequence is called *time series*.

Among neural networks, Recurrent Neural Networks (RNNs) [108, 42, 41] represent a widely known class of neural networks suitable for learning in sequential domains. In addition to feed-forward connections, such architectures are characterized by feedback connections between neurons (i.e., recurrent connections). Through the recurrent connections, they implement a dynamical system with a *state* that represents the memory of the past input collected by the network. Recently, RNNs obtained excellent performance in many application fields such as, time-series prediction [23, 129], computer vision [74, 75], language modelling [132, 183], machine translation [29, 185] and speech recognition [73, 159] providing extremely flexible solutions in processing different kind of complex sequences. Such models are typically trained by gradient descent algorithms. Typical approaches in designing RNNs are focused on modelling long short-term temporal dependencies by proposing

training solutions to improve gradient descent algorithms (by addressing vanishing and exploding gradient issues [15, 146]) or by designing architectures able to learn how much past information preserve inside the network (i.e., *gated RNNs* [88, 29, 31]).

An efficient class of RNN within Reservoir Computing (RC) framework is represented by Echo State Networks (ESNs). ESNs are composed by a non-linear recurrent layer called *reservoir* and a linear output layer called *readout*. The ESNs are *randomized* RNNs in which the recurrent layer is randomly initialized in order to satisfy a stability condition on the dynamical system [191, 99, 213, 52], while the readout is trained by using direct methods. Since in ESN architectures the output layer is the only trained part of the network, the learning algorithm results much more efficient than typical RNN approaches in which all weights are trained. The success of ESNs obtained in benchmark applications [98, 99] contributed to arouse the interest of neural networks' community (see [200, 121] for reviews). As for general RNNs, also in RC field the analysis of long short-term dependencies and memory capacity are relevant and studied for instance through the use of *leaky integrators* [100, 96], learning algorithms [44, 45] and minimum complexity architectures [154, 155].

In the field of time-series processing, the temporal data is typically characterized by different time scales. In this case, a time series can be seen as a composition of signals characterized by different frequency components. Therefore, the quality of the model's performance is related to the capacity in representing multiple time-scales dynamics. Accordingly, within RC literature, some works propose recurrent architectures designed to model multiple time-scales dynamics [206, 211, 168, 89, 109, 139, 151, 17].

Despite the good results obtained on artificial tasks in RC field, the design of ESNs for practical real-world applications can be difficult [150]. In particular, the study of RC solutions for more complex tasks is an open research issue [194, 164]. Moreover, there is the need for more investigations in challenging tasks characterized by high dimensional sequences and multiple time-scales dynamics such as text processing [184] and polyphonic music [23].

Concerning the design of novel architectures aimed at enriching the inner representation of the input information, in the last decade, the study of *deep neural networks* has stimulated the interest of the research community. Based on a hierarchy of multiple layers, such models provide a distributed, hierarchical and compositional representation of the input information allowing to address challenging real-world

problems [73, 134, 185, 110, 72]. In particular, studies on *deep RNNs* opened a way to develop novel models able to learn hierarchical temporal representations from signals characterized by multiple time-scales dynamics [167, 83, 95, 82]. Some works in literature aimed at obtaining multiple time-scales dynamics in layered RNNs by forcing different time delays [83, 30] or subsampling between layers [74]. Previous works on hierarchical organizations in the Reservoir Computing (RC) area [121, 200] are based on gradient descent approaches [95] or pipeline of progressively trained ESN modules [193].

Although deep RNNs obtained good results in some applications area (see above), there are still fundamental open issues about the architectural aspects that deserve further investigations. Specifically, as a general issue underlying this work, there is the need to understand whether and to what extent layering in recurrent architecture deserves to be used. This can be investigated by studying and analysing the intrinsic role of layering in RNNs. We can hence summarize the main open issues in deep RNNs in the following points:

- The observation that stacking layers of recurrent units inherently creates different time-scales dynamics at different layers [82, 144], and therefore a hierarchical representation of the temporal information *per se*, deserves to be investigated and analyzed. The same can be said about the impact of layering on the model's performance.

- Quantitative and qualitative measures are needed to study the intrinsic characterization of layering in deep RNNs.

- The training of deep architecture and recurrent neural networks can be very difficult. Indeed, deep RNNs implements few layers (< 10) in state-of-the-art applications.

- The training of all recurrent weights in a deep recurrent architecture can be very expensive.

- Currently, the number of layers in deep recurrent architectures is determined by *trial and error* approaches resulting in expensive procedures. Can the analysis help us to find better design solutions?

- Concerning hierarchies of recurrent layers in RC computing, the study on the effect of different parameters of RC models on the dynamical behavior and

the performance of deep RNNs, without the supervised training of hidden layers, is missing in literature.

Given such considerations, it emerges that the combination of RNN models, RC paradigm and Deep Learning approaches, can provide a novel class of deep recurrent architectures to address open issues related to characterization, efficiency and design of deep RNNs. Overall, these aspects constitute the subject of the thesis.

1.2 Objectives of the Thesis

The main objective of the thesis is to introduce a novel class of deep RNNs within the RC paradigm for the analysis and the developing of efficient deep recurrent architectures able to process signals characterized by multiple time-scales dynamics. The first aim is to analyze the role of layering in deep RNNs by studying how the hyper parameters and the architecture topologies affect the intrinsic ability of a multi-layered architecture in developing distributed and hierarchical temporal features. In order to analyze and study such architectures, we intend to define quantitative and qualitative measures based on temporal differentiation, entropy, memory capacity and frequency analysis tools. After that, we aim to define an automatic design algorithm to determine the number of layers of a deep RNN based on frequency analysis tools.

A secondary objective is to provide efficient solutions for deep RNNs based on hierarchical reservoir organization able to compete with the state-of-the-art approaches in challenging real-world tasks characterized by multiple time-scales dynamics.

1.3 Contributions of the Thesis

The main contributions of the thesis are summarized in the following.

Deep Reservoir Computing

We explicitly extended the RC framework to Deep Learning paradigm introducing a novel class of randomized deep RNN architectures. Thereby, we investigated how the RC parameters and the architecture topology influence the model performance and the temporal features developed by recurrent architectures. First, we introduced the *DeepESN* model characterized by a hierarchy of randomized recurrent layers.

After that, for the analysis purpose, we compared DeepESN with standard ESN (the shallow counterpart) and other architectural variants, namely *DeepESN Input to All* (DeepESN-IA) and *GroupedESN*. Our analysis highlighted that hierarchical layering intrinsically improves the temporal representation in terms of multiple time-scales differentiation, memory capacity and quantity of entropy in deep RNNs.

Analysis of deep RNNs through Deep Echo State Networks

The main model introduced in this thesis (i.e., DeepESN) is composed by a hierarchy of non-linear recurrent layers and a linear output layer. The recurrent layers are randomly initialized and left untrained, while the output layer is trained by direct approaches. The output is computed through a weighted sum of the states coming from all recurrent layers in the architecture. In such a way, the training phase allows the model to weights the contributions of the multiple time-scales dynamics involved in the architecture. Moreover, we introduced a pre-training phase for deep RNNs based on Intrinsic Plasticity (IP) algorithm. Through DeepESN approaches, we defined qualitatively and quantitatively measures to analyze the intrinsic characterizations of layering in deep RNNs. The analysis showed that layering is intrinsically able to develop a multiple time-scales differentiation among the architecture without training recurrent weights and even with fixed RC parameters. Moreover, it is possible to enhance the time-scales differentiation, the memory capacity and the quantity of entropy by varying RC parameters over the layers or by applying the IP for deep RNNs.

Analysis of deep RNNs through Linear Deep Echo State Networks

The Linear Deep Echo State Network (L-DeepESN) model is a DeepESN architecture that implements linear activation functions. Therefore, it is composed by a hierarchy of linear recurrent layers. The learning is performed as in DeepESN. Through L-DeepESN architecture, we performed frequency analysis on the network's states to study the structure of the temporal features that naturally emerges in layered RNNs. The use of linear activations clarifies the analysis avoiding harmonic distortion in the frequency spectrum. Moreover, linear recurrent layers allowed us to find an algebraic characterization of the filtering effect progressively performed by layered RNNs. The L-DeepESN approach outperforms the shallow counterpart (i.e., an L-DeepESN with one layer) on recent challenging versions of time-series prediction tasks characterized by multiple superimposed sinusoids. Overall, the

spectral analysis revealed that the higher frequency components are progressively filtered in the higher levels of the network. Thereby, the deep recurrent architecture develops a multiple frequency representation of the input signal ordered along the layered recurrent architecture. Therefore, it turns out that layering in deep RNNs enables to intrinsically develop hierarchical and distributed temporal features even without learning and also without the use of non-linear projections.

Design of deep RNNs through Deep Echo State Networks

We proposed a novel approach to analyze and design efficient deep RNNs through signal processing tools. In particular, we defined an automatic design algorithm to determine the number of layers of a DeepESN avoiding to perform an expensive trial and error procedure. The proposed algorithm also provides an efficient tool to analyze the kind of filtering effect progressively performed in the depth of a layered recurrent architecture over the input signals. In addition to the typical low-pass filtering effect, our empirical analysis showed that it is also possible to produce an high-pass filtering effect by using the IP algorithm. Our design approach allowed for the first time in literature to obtain competitive results with the state of the art in challenging real-world tasks using deep RNN composed by tens of recurrent layers (> 30).

Deep Echo State Networks for Real-world applications

First, we developed an original medical system for human balance assessment for RC paradigm, comparing the ESN approach with basic neural networks and with the DeepESN model. Then, we developed another medical application for the diagnosis of Parkinson's diseases comparing DeepESN with the shallow counterpart. Finally, we evaluated DeepESN on challenging high-dimensional time-series tasks. The DeepESN is evaluated in all considered tasks for different purposes. In general, the DeepESN is compared to the shallow counterpart in order to assess the advantage of layering in terms of performance. In speech recognition task, DeepESN is compared with other RC approaches. Whereas, on polyphonic music tasks the DeepESN is compared with fully trained RNNs typically used in DL community. In the considered tasks, the DeepESN model always outperformed the other considered RC approaches. Moreover, in most cases, DeepESN outperformed fully trained and gated RNNs on time-series prediction tasks based on polyphonic music datasets obtaining better training time performances. Overall, the proposed approaches

allowed us to design efficient deep RNN models competitive with the state of the art on challenging real-world tasks.

1.4 Plan of the Thesis

This thesis is organized in 6 Chapters.

In Chapter 1, we introduce the most relevant aspects of the thesis concerning motivations, objectives, contributions, plan and origins of chapters.

In Chapter 2, we present the basic research aspects from RNN and RC background which are of specific interest for this thesis. In Section 2.1, we briefly describe the main aspects regarding machine learning and neural networks approaches. In Section 2.2, we present feed forward neural networks for flat domains. In Section 2.3, we introduce the typical RNN approaches for sequence processing. In Section 2.4, we introduce the randomized RNNs with a particular interests to ESN model within the RC framework. Finally in Sections 2.5, 2.6 and 2.6.6, we present the recent studied aspects of Deep Learning paradigm with a particular focus on deep RNN models and then we discuss the open issues about the design and the analysis of deep recurrent architectures.

In Chapter 3, we introduce a novel class of deep RNNs that we called Deep Reservoir Computing defining and analysing randomized deep RNNs characterized by many recurrent layers connected to each other and implemented within the RC paradigm. In particular, in Section 3.2, we introduce the DeepESN model. In Section 3.2.1, we define the baselines (i.e., DeepESN-IA and GroupedESN models) in order to analyze and compare different kind of randomized deep RNN architectures. Accordingly, in Section 3.3, we study the intrinsic temporal characterization of deep RNNs using deep RC models. Moreover, in Section 3.4, we define the L-DeepESN model for the analysis and the study of the characterization of the temporal features in layered recurrent architectures.

In Chapter 4, we introduce a design approach to buid up efficient deep recurrent architectures and to analyze the depth of deep RNNs. In Section 4.2, we define the proposed design algorithm, and then we evaluate it on a controlled scenario. For the sake of the organization of this thesis, the evaluation of such approach on

real-world applications is described in Chapter 5.

In Chapter 5, we introduce a series of tasks based on RC paradigm and DeepESN model. First, in Section 5.1 we develop a medical system for clinical analysis of human balance assessment based on RC models, comparing the ESN model with basic neural networks approaches. Then, in Section 5.1.4 we compare the ESN approach with the DeepESN model on the baseline exercise of balance assessment task. In Section 5.2, we present a method for the diagnosis of Parkinson's disease based on DeepESN. In Section 5.3 we assessed and analyzed the DeepESN model on challenging real-world tasks characterized by high-dimensional time series concerning speech recognition and polyphonic music tasks. On such tasks, we evaluated and analyzed the design approaches for DeepESNs defined in Section 4. In particular, we compare the DeepESN model with other RC approaches on the speech recognition task in Section 5.3.2. Finally, we compare the DeepESN approach with typical fully trained RNNs on 4 polyphonic music tasks in Section 5.4.

Finally, Chapter 6 draws the conclusions of the thesis.

1.5 Origin of the Chapters

Most research studies introduced in this thesis are already published on conference proceedings and as journal papers:

- the DeepESN, DeepESN-IA and GroupedESN models are introduced in the journal paper [61], which extends the preliminary work published in [54]. In particular, the Chapter 3 is based on works presented in [61, 63];
- the L-DeepESN model presented in Section 3.4.1 is published in [63];
- Chapter 4 is based on the journal paper presented in [62];
- the application concerning the diagnosis of Parkinson's presented in Section 5.2 is published in [60];
- the system for the automatic berg balance scale estimation presented in Section 5.1 is published in [6, 64, 65];

- the experimental assessments on speech recognition and polyphonic music tasks described in Section 5.3.1 are published in [62, 59].

Deep Learning Background

In this Chapter we review the main aspects concerning the Recurrent Neural Networks field with a particular interest in Reservoir Computing paradigm and Deep Learning background. In Sections 2.1 and 2.2 we briefly introduce the Machine Learning field and Neural Networks for flat data processing. In Section 2.3, we review typical Recurrent Neural Network models for sequence processing. In Section 2.4, we introduce randomized Recurrent Neural Networks with a particular interest in Echo State Network models within Reservoir Computing paradigm. In Section 2.5, we summarize the main aspects of Deep Learning models. Finally, in Sections 2.6 and 2.6.6, we describe the principal open issues in Deep Learning with a particular focus on deep recurrent architectures.

2.1 Machine Learning and Neural Networks

Machine Learning (ML) is a sub field of Artificial Intelligence and it deals with the implementation of systems and algorithms that are based on observations. ML concerns the analysis and development of algorithms that can learn from data in order to estimate and make predictions [130]. Learning can take place by capturing features of interest from examples, data structures or sensors, to analyze and evaluate the relationships between the observed variables. ML models are developed in order to fit the data and to generalize instead of following static program instructions.

In ML, the data is typically composed by observations, in which each observation is a fixed sized vector of variables. The typology of data can be divided into *unstructured* and *structured*. An unstructured domain is characterized by flat collections composed by a set of observations without dependencies between them, while structured domain is characterized by collections of observations that have relationships between them such as sequences, trees and graphs.

Typically, a model can be learned in *unsupervised* or in *supervised* way. In unsupervised learning, the model is trained in order to discover how data is organized. Some instances of unsupervised learning tasks are clustering, dimensionality reduction and data visualization. In the case of supervised learning, the model is trained to infer a function from a training set. The training set is a labeled data collection that represents the input-output relations to be learned for the model.

In supervised learning there are two common tasks, *classification* and *regression*. In classification tasks, the target output is a vector that represents the class to which the input belongs, while in regression tasks, the target output is a vector of real values of an unknown continuous function of the related input. Let $\mathcal{U}$ be the input space and $\mathcal{Y}$ the output space. The supervised learning task consists in finding the best approximation of the unknown function $f : \mathcal{U} \to \mathcal{Y}$, on the basis of a training set $T_{\text{train}} = \{(\mathbf{u}(i), \mathbf{y}(i))\}_{i=1}^{N_T}$, where N_T is the number of available training examples, $\mathbf{u}(i)$ is the i-th input pattern and $\mathbf{y}(i) = f(\mathbf{u}(i))$ is the i-th target pattern. Supervised learning often consists in finding the free parameters $\mathbf{w}$ of *hypothesis* function $h_w : \mathcal{U} \to \mathcal{Y}$ which approximates the function f. The hypothesis space of the learning model is $H = \{h_{\mathbf{w}} : \mathcal{U} \to \mathcal{Y} | \mathbf{w} \in \mathbf{W}\}$, where $\mathbf{W}$ is the parameters space. The computed function from the learning machine is $\mathbf{y} = h_{\mathbf{w}}(\mathbf{u})$. The *loss function* $L : \mathcal{U} \times \mathcal{U} \to \mathbb{R}$ is used to measure the distance between actual and desired value of the computed function. The learning goal consists in minimizing the *risk functional*

$$R(\mathbf{w}) = \int L(\mathbf{y}, h_{\mathbf{w}}(\mathbf{u})) dP_{\mathbf{u},\mathbf{y}}(\mathbf{u}, \mathbf{y}), \tag{2.1}$$

where $P_{\mathbf{u},\mathbf{y}}(\mathbf{u}, \mathbf{y})$ is the joint cumulative distribution function for $\mathbf{u}$ and $\mathbf{y}$. Typically, the distribution of $P_{\mathbf{u},\mathbf{y}}(\mathbf{u}, \mathbf{y})$ is unknown, therefore, minimizing directly $R(\mathbf{w})$ could be not possible. Since only the training set T_{train} is known, a very common method, based on the *principle of empirical risk minimization*, is minimizing the loss function L on the training set examples. Thus, the learning algorithm of the model consists in minimizing the *empirical risk* $R_{\text{emp}}(\mathbf{w})$ which is an approximation of the risk functional in Equation 2.1:

$$R_{\text{emp}}(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} L(\mathbf{y}(i), h_{\mathbf{w}}(\mathbf{u}(i))). \tag{2.2}$$

The relationship between $R(\mathbf{w})$ and $R_{\text{emp}}(\mathbf{w})$ is studied by the *Statistical Learning Theory* [199, 198].

The class of ML that we mainly take into account in this thesis is artificial Neural Networks (NNs). Artificial NNs are a family of learning models inspired by biological NNs [79]. NNs can be defined by a directed graph, where the vertices represent the neurons and the edges represent the connections between them. Each neuron is associated with an activation function and each connection has a weight value. Omitting the bias terms in the following formulas for the ease of notation, the output value $y \in \mathbb{R}$ of a neuron is computed by the following equation:

$$y = f(\mathbf{w}^\intercal \mathbf{u}) = f(\sum_{i=1}^{N_U} \mathbf{w}(i)\, \mathbf{u}(i)), \tag{2.3}$$

where f is the activation function of the neuron joined with N_U incoming edges, $\mathbb{R}^{N_U}$ is the input space, $\mathbf{u} \in \mathbb{R}^{N_U}$ is the input vector and $\mathbf{w} \in \mathbb{R}^{N_U}$ is the weight vector of the incoming edges. Figure 2.1 shows a typical architecture of an artificial neuron connected to an input layer. The neuron is fed from a pool of input neurons.

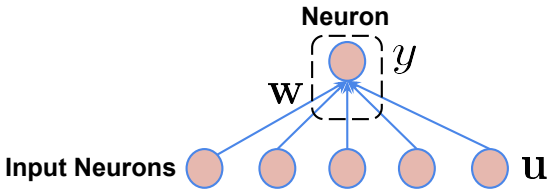


Figure 2.1: A typical architecture of a neuron in artificial NNs class.

Thereby, the output computation is divided in two steps, the former is a linear combination between the vector of weights $\mathbf{w}$ and the input vector $\mathbf{u}$ and the latter is an activation function that typically computes a non-linear transformation. A pool of N_Y neurons is called *layer*. Figure 2.2 shows the architecture of a layer. The computation of the output vector $\mathbf{y} \in \mathbb{R}^{N_Y}$ is performed as follows:

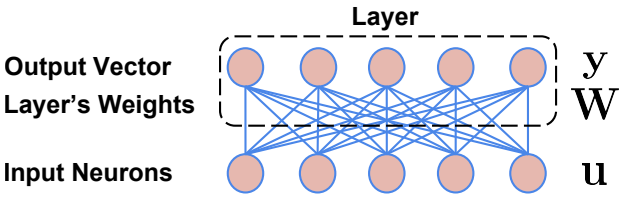


Figure 2.2: A typical architecture of a layer in artificial NNs class.

$$\mathbf{y} = f(\mathbf{W}\mathbf{u}) = \begin{bmatrix} f(\sum_{i=1}^{N_U} \mathbf{W}(1,i)\,\mathbf{u}(i)) \\ \vdots \\ f(\sum_{i=1}^{N_U} \mathbf{W}(N_Y,i)\,\mathbf{u}(i)) \end{bmatrix}, \tag{2.4}$$

where f is the activation function, $\mathbf{u} \in \mathbb{R}^{N_U}$ is the input vector and $\mathbf{W} \in \mathbb{R}^{N_Y \times N_U}$ is the matrix of weight values that represents the incoming connections of the layer. The layer is called *hidden layer* if it is positioned between other two other layers. If the layer computes the output of the network, it is called *output layer*. A neural network architecture can be composed by many layers connected between them.

2.2 Feed-forward Neural Networks

Feed-forward Neural Networks (FFNNs) are a class of NNs in which the information is propagated from the input layers towards the output layer without feedback connections. They are able to process flat data such as fixed sized vectors of variables. *Multi-layer Perceptrons* (MLPs) [79] are a typical class of FFNNs. They are composed by a stack of non-linear hidden layers which are fully connected by feed-forward connections.

Figure 2.3 shows the architecture of an MLP with a single non-linear hidden layer. Omitting the bias terms for the ease of notation, the output of the network

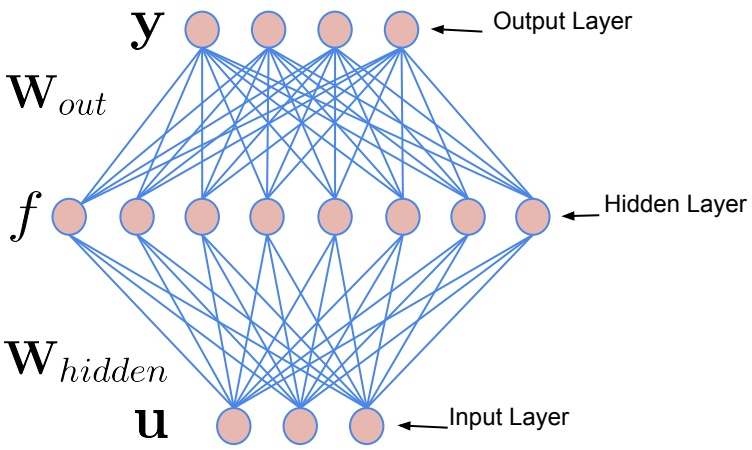


Figure 2.3: Architecture of MLPs with a single non-linear hidden layer.

is computed as follows:

$$\mathbf{y} = \mathbf{W}_{\text{out}} f(\mathbf{W}_{\text{hidden}} \mathbf{u}), \tag{2.5}$$

where f is the activation function (typically *sigmoidal functions*), $\mathbf{u} \in \mathbb{R}^{N_U}$ is the input pattern, $\mathbf{y} \in \mathbb{R}^{N_Y}$ is the output pattern, $\mathbf{W}_{\text{hidden}} \in \mathbb{R}^{N_R \times N_U}$ is the hidden weight matrix (where N_R is the number of hidden units) and $\mathbf{W}_{\text{out}} \in \mathbb{R}^{N_Y \times N_R}$ is the output weight matrix. MLPs are typically trained with gradient-based techniques such as *BackPropagation* (BP) and *Stochastic Gradient Descent* (SGD) [158, 22, 21, 107]. An MLP with at least one non-linear hidden layer is featured by universal approximation capabilities [37, 91]. Although MLPs can approximate a wide variety of continuous functions, these theoretical results does not provide a procedure to learn the parameters of the model.

2.3 Recurrent Neural Networks

In this Chapter we introduce typical Recurrent Neural Network models for sequence processing. In Section 2.3.1, we describe the causal and stationary transductions on sequence domains that characterize the computation process of Recurrent Neural Networks. Then, in Section 2.3.2 we introduce the simple Recurrent Neural Network model. Finally, in Section 2.3.3 we review the most important characteristics of gated Recurrent Neural Networks.

2.3.1 Causal Transactions for Sequence Processing

Here we introduce a typical sequence processing approach implemented by causal and stationary transductions on sequence domains. In the following, we denote as $\mathbf{s}(n) = [\mathbf{s}(1), ..., \mathbf{s}(n)] \in \mathcal{U}^*$ a sequence of length $n > 1$, where $\mathcal{U}$ is the input label space. The element $\mathbf{s}(1)$ represents the oldest entry and $\mathbf{s}(n)$ is the most recent one.

A transduction on sequence domains $\mathcal{T} : \mathcal{U}^* \to \mathcal{Y}^*$ can be computed by recursively applying encoding transduction $\mathcal{T}_{\text{enc}} : \mathcal{U}^* \to \mathcal{X}^*$ and output transduction $\mathcal{T}_{\text{out}} : \mathcal{X} \to \mathcal{Y}$ where $\mathcal{X}$ and $\mathcal{Y}$ are state and output label spaces. In particular, the encoding transduction is implemented through the *local transition function* $\tau : \mathcal{U} \times \mathcal{X} \to \mathcal{X}$ defined for $t = 1, ..., n$ as follows:

$$\mathbf{x}(t) = \tau(\mathbf{u}(t), \mathbf{x}(t-1)), \tag{2.6}$$

where an initial state $\mathbf{x}(0) \in \mathcal{X}$ is defined. Now we can define the *global state*

transition function $\hat{\tau} : \mathcal{U}^* \times \mathcal{X} \to \mathcal{X}$ as the reflexive and transitive closure of τ. Function $\hat{\tau}$ corresponds to the extension of τ to paths in finite automata [90]. Accordingly, function $\hat{\tau}$ is defined as:

$$\hat{\tau}(\mathbf{s}(\mathbf{u}), \mathbf{x}(0)) = \begin{cases} \mathbf{x}(0), & \text{if } \mathbf{s}(\mathbf{u}) = [] \\ \tau(\mathbf{u}(n), \hat{\tau}([\mathbf{u}(1), ..., \mathbf{u}(n-1)], \mathbf{x}(0))) & \text{if } \mathbf{s}(\mathbf{u}) = [\mathbf{u}(1), ..., \mathbf{u}(n)], \end{cases} \tag{2.7}$$

where $\hat{\tau}(\mathbf{s}(\mathbf{u}), \mathbf{x}(0))$ is obtained by recursively applying the state transition τ to input sequence $\mathbf{s}(\mathbf{u})$. Thereby, given an initial state $\mathbf{x}(0) \in \mathcal{X}$, $\mathcal{T}_{\text{enc}}(\mathbf{s}(\mathbf{u})) = \hat{\tau}(\mathbf{s}(\mathbf{u}), \mathbf{x}(0))$. Then, the output transition $\mathcal{T}_{\text{out}}$ is computed by locally applying the output function $g_{\text{out}} : \mathcal{X} \to \mathcal{Y}$:

$$\mathbf{y}(t) = g_{\text{out}}(\mathbf{x}(t)), \tag{2.8}$$

where $\mathbf{y}(t) \in \mathcal{Y}$ is the t-th element of the output sequence $\mathbf{y}(\mathbf{s}) \in \mathcal{Y}^*$. Equation 2.8 is used in the case of *sequence-to-sequence* transductions. In the case of *sequence-to-element* transductions the output sequences degenerate into a single element $\mathbf{y}(\mathbf{s}) \in \mathcal{Y}$. This can be obtained defining a state mapping function $\chi : \mathcal{X}^* \to \mathcal{X}$. A typical approach is referred as *root state mapping* and it consists in selecting the last element of the state $\mathbf{s}(\mathbf{x}) = [\mathbf{x}(1), ..., \mathbf{x}(n)] \in \mathcal{X}^*$:

$$\chi(\mathbf{s}(\mathbf{x})) = \mathbf{x}(n). \tag{2.9}$$

Another state mapping function that we consider in the following is called *mean state mapping*. It is obtained by computing over the length n of the state $\mathbf{s}(\mathbf{x}) = [\mathbf{x}(1), ..., \mathbf{x}(n)] \in \mathcal{X}^*$:

$$\chi(\mathbf{s}(\mathbf{x})) = \frac{\sum_{t=1}^{n} \mathbf{x}(t)}{n}. \tag{2.10}$$

2.3.2 Simple Recurrent Neural Networks

Recurrent Neural Networks (RNNs) [108, 42, 41] are a class of NNs which implement feedback connections between neurons. They are an extension of NNs for sequence processing. In particular, they are suitable to process sequences of observations such as time series. RNNs can compute causal (the calculation depends only on the past), stationary (the system does not change over time) and adaptive (all weights are learned) transductions on sequence domains. Recurrent connections allow the neurons to store memory by means of a temporal context that represents the encoding of the input history. This provides a system that potentially can keep

the entire information of an arbitrary period instead of having a fixed time length.

Figure 2.4 shows the architecture of a simple RNN (SRN).

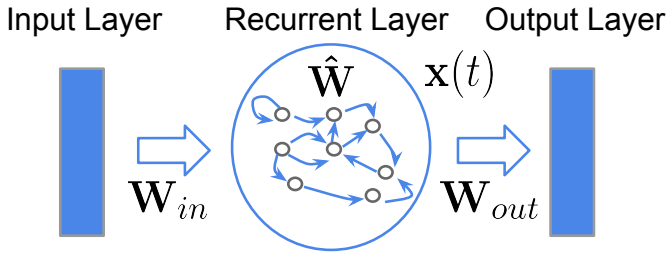


Figure 2.4: Architecture of SRN model.

Omitting the bias terms for the ease of notation, the state of the network $\mathbf{x}(t) \in \mathbb{R}^{N_R}$ at time $t > 0$ is computed as follows:

$$\mathbf{x}(t) = \tau(\mathbf{u}(t), \mathbf{x}(t-1)) = f(\mathbf{W}_{\text{in}}\mathbf{u}(t) + \hat{\mathbf{W}}\mathbf{x}(t-1)), \tag{2.11}$$

where $\mathbf{u}(t) \in \mathbb{R}^{N_U}$ is the input pattern at time t, f is usually a non-linear activation function of sigmoidal type, $\mathbf{W}_{\text{in}} \in \mathbb{R}^{N_R \times N_U}$ is the matrix of the input weights, $\hat{\mathbf{W}} \in \mathbb{R}^{N_R \times N_R}$ is the matrix of recurrent weights and $\mathbb{R}^{N_R}$ is the state space. The initial state $\mathbf{x}(0) \in \mathbb{R}^{N_R}$ is typically defined as the null vector $\mathbf{0}$. The output of the network at time-step t is computed as follows:

$$\mathbf{y}(t) = f_{\text{out}}(\mathbf{W}_{\text{out}}\mathbf{x}(t)), \tag{2.12}$$

where $\mathbb{R}^{N_Y}$ is the output space, $\mathbf{y}(t) \in \mathbb{R}^{N_Y}$ is the output pattern at time-step t, $\mathbf{W}_{\text{out}}$ the matrix of output weights and f_{out} can be a linear or non-linear activation function. Training is performed using standard algorithms in the field of RNNs [196] [79], such as *BackPropagation through time* (BPTT) [203].

2.3.3 Gated Recurrent Neural Networks

RNNs can have problems to manage long term dependencies because the gradients propagated over many time-steps tend to vanish or explode [146]. A way to manage long term dependencies is to introduce in the models leaky integrator units [100, 209]. Indeed, the leaky parameter can control the contribution of the past input in the state dynamic. However, the leaky parameter is fixed and it is not learned by data. A solution in which the integrator is not fixed consists in using trainable weights in a gated self-loop. A class of NNs that implements a sort of adaptive

integrator units (i.e., gates) is called *Long Short-Term Memory* (LSTM) [88]. Figure 2.5 shows the architecture of an LSTM cell.

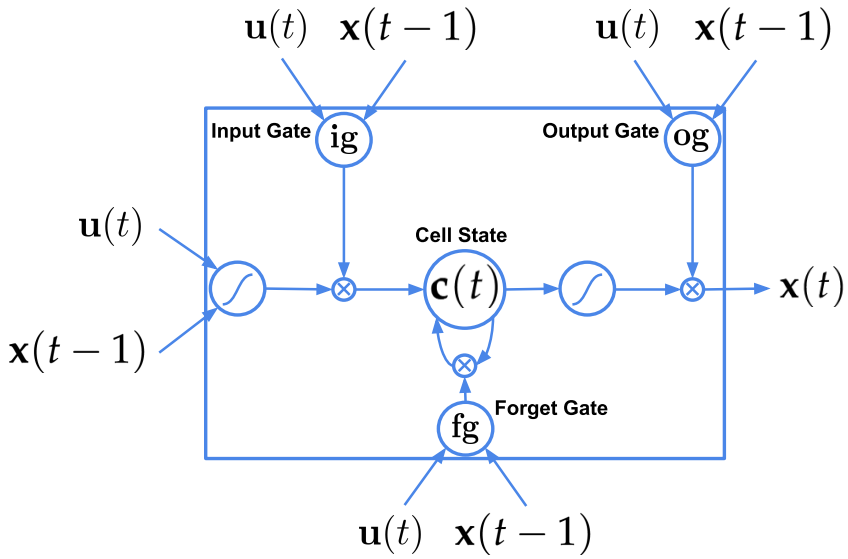


Figure 2.5: A typical architecture of LSTM cell.

In the following equations, we omit the bias terms for the ease of notation. The LSTM cell is composed by an *input gate* $\mathbf{ig}(t)$, a *forget gate* $\mathbf{fg}(t)$, an *output gate* $\mathbf{og}(t)$, the state of the *cell* $\mathbf{c}(t)$ and the output of the cell $\mathbf{x}(t)$[1]. Overall, the LSTM architecture implements two state variables, $\mathbf{c}(t)$ and $\mathbf{x}(t)$. In particular, since $\mathbf{x}(t)$ is passed to the output layer or more generally to the next layer, it can be considered the state of the LSTM layer, while, the cell state $\mathbf{c}(t)$ is visible only between the cells of the layer. In formulas the LSTM cell is defined by the following equations:

$$\mathbf{ig}(t) = \sigma(\mathbf{W}^{\text{ig}}{}_{\text{in}}\mathbf{u}(t) + \hat{\mathbf{W}}^{\text{ig}}\mathbf{x}(t-1)), \tag{2.13}$$

$$\mathbf{fg}(t) = \sigma(\mathbf{W}^{\text{fg}}{}_{\text{in}}\mathbf{u}(t) + \hat{\mathbf{W}}^{\text{fg}}\mathbf{x}(t-1)), \tag{2.14}$$

$$\mathbf{c}(t) = \mathbf{fg}(t) \odot \mathbf{c}(t-1) + \mathbf{ig}(t) \odot tanh(\mathbf{W}_{\text{in}}\mathbf{u}(t) + \hat{\mathbf{W}}\mathbf{x}(t-1)), \tag{2.15}$$

$$\mathbf{og}(t) = \sigma(\mathbf{W}^{\text{og}}{}_{\text{in}}\mathbf{u}(t) + \hat{\mathbf{W}}^{\text{og}}\mathbf{x}(t-1)), \tag{2.16}$$

$$\mathbf{x}(t) = tanh(\mathbf{c}(t)) \odot \mathbf{og}(t), \tag{2.17}$$

where $\odot$ is the multiplication operation between vectors. The matrices $\mathbf{W}_{\text{in}}$ and $\hat{\mathbf{W}}$ represent the input and the recurrent weights of the network. Moreover, the matrices $\mathbf{W}^{\text{ig}}$, $\mathbf{W}^{\text{fg}}$ and $\mathbf{W}^{\text{og}}$ represent the input weights of input, forget and output

[1]We used different variable names with respect the typical LSTM notation to keep the same notation used in this thesis for RNNs in Section 2.3.2.

gate, respectively. Finally, the matrices $\hat{\mathbf{W}}^{\text{ig}}$, $\hat{\mathbf{W}}^{\text{fg}}$ and $\hat{\mathbf{W}}^{\text{og}}$ represent the recurrent weights of input, forget and output gate, respectively.

The input gate $\mathbf{ig}(t)$ and the forget gate $\mathbf{fg}(t)$ regulate the contribution of the current input and the past of the cell state. While, the output gate $\mathbf{og}(t)$ adjusts the passage of information between the cell state and the cell output. From Equations 2.15 and 2.17 we can see that the update of the state variables, $\mathbf{c}(t)$ and $\mathbf{x}(t)$ follows the state-transition mechanism (i.e., the new state depends on the previous state and the new input).

The training in LSTMs is performed as in standard RNNs through BPTT algorithms, however, the gated self-loop of LSTM cells allow to attenuate the vanishing gradient problem [88, 15]. Literature studies showed LSTMs learning long-term dependencies more easily than SRNs first on artificial tasks [15, 88, 87] and then also on challenging sequence processing tasks [73, 185].

The downside of LSTMs is that they have a big number of parameters, since they implement input and recurrent weights for each gate. In order to simplify gated architectures without losing the ability to learn long-term dependencies, further works introduced Gated Recurrent Units (GRUs) [29, 31]. Figure 2.6 shows the architecture of a GRU cell.

In particular, they reduce the number of state variables (from 2 to 1) and the number of gates (from 3 to 2). Thereby, the GRU architecture implement an update gate $\mathbf{z}$ and a reset gate $\mathbf{r}$ defined respectively as:

$$\mathbf{z}(t) = \sigma(\mathbf{W}^{z}_{\text{in}}\mathbf{u}(t) + \hat{\mathbf{W}}^{z}\mathbf{x}(t-1)), \tag{2.18}$$

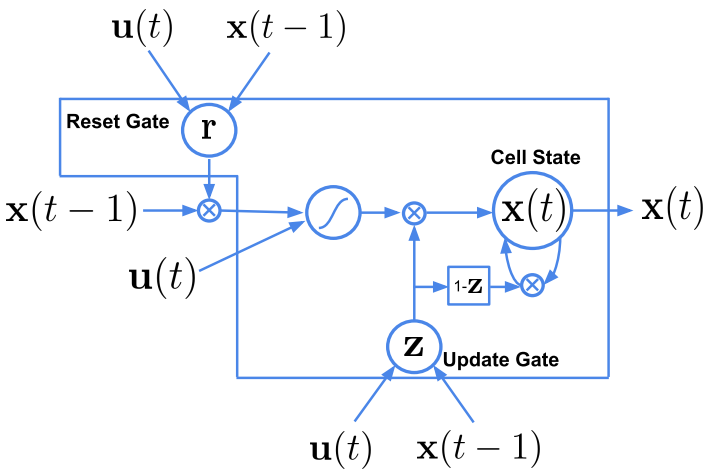


Figure 2.6: A typical architecture of LSTM cell.

$$\mathbf{r}(t) = \sigma(\mathbf{W}_{\text{in}}^{r}\mathbf{u}(t) + \hat{\mathbf{W}}^{r}\mathbf{x}(t-1)). \tag{2.19}$$

Moreover, the computation of the state $\mathbf{x}(t)$ at time-step t is performed as follows:

$$\mathbf{x}(t) = (1 - \mathbf{z}(t)) \odot \mathbf{x}(t-1) + \mathbf{z}(t) \odot \sigma(\mathbf{W}_{\text{in}}\mathbf{u}(t) + \hat{\mathbf{W}}(\mathbf{r}(t) \odot \mathbf{x}(t-1))). \tag{2.20}$$

Similarly to leaky integrator units (see parameter a in Equation 2.21), the update gate $\mathbf{z}$ controls the velocity of the dynamics of the state. Moreover, the reset state $\mathbf{r}$ determines the quantity of the past state to discard.

2.4 Randomized Recurrent Neural Networks

Randomized NNs are characterized by the use of random methods applied to NNs in which only some parts of the architecture are trained [50, 163, 68]. Typically, they are composed by untrained hidden layers randomly initialized and used to project the input patterns into a high dimensional feature space in a non-linear way. The training is performed on the output layer (i.e., *readout*) exploiting the hidden layers' representation. Pioneering works on Perceptrons [18, 157, 156] explore architectures composed by layers with random weights. In particular, in some versions of Perceptron, the connections between retina sensory units (input layer) and associator units (hidden layer) are randomly initialized while the last layer is iteratively adapted with the Perceptron learning rule. The use of randomized connections between layers in NNs is exploited on a number of early works [48, 169, 1]. In particular in [169], it is investigated the performance of an MLP with one hidden layer in which the values of the weights are selected randomly from a uniform distribution in the range $[-1, +1]$ and the output layer is trained using direct methods. This work shows that it is possible to achieve good classification results without training all the network's weights. Another method that uses a randomization approach is called random-vector functional-link Net [142]. In this model, the representation developed by the hidden layer is formed by concatenating the output of the random projection with the input vector. Thereby, the readout can also directly exploit the input information. Other architectures that exploit non-linear random projections and direct learning methods in different forms of FFNNs are Radial Basis Functions with random centers [24], Extreme Learning Machines [92], the No-Prop algorithm [205] and Stochastic Configuration Networks [202].

In randomized RNNs, the paradigm is extended to sequence processing [68].

In this case, recurrent layers are left untrained and randomly initialized. The recurrent part of the network encodes the input in a state-based memory exploited by the readout to compute the output. In the case of discrete-time, the encoding process performed by a recurrent layer can be seen as a non-autonomous dynamical system described by means of iterated functions. One of the fundamental issues in randomized RNNs is to ensure the stability of the encoding process. This can be obtained by imposing contractive properties in the initialization of recurrent layers. A well known research field that studies these aspects regards randomized RNNs implemented within Reservoir Computing (RC) [200, 121] paradigm. In the following, we focus on RC framework with a particular interest to the Echo State Network (ESN) [99] model.

2.4.1 Reservoir Computing and Echo State Networks

RC [200, 121] is an efficient paradigm for the design and implementation of RNNs. The model is composed by two components, a dynamic layer with recurrent connections called *reservoir* and a linear output layer called *readout*. Usually, the reservoir contains a large number of recurrent units, connected to each other in a sparse manner. The recurrent weights of the reservoirs are randomly initialized according to specific criteria and then left untrained [98]. In particular, small weights initializations characterize a contractive dynamics of the network state [77, 189, 191, 192]. The readout is composed by linear units and it is trained by direct methods [200, 121]. In these models, the computation is characterized by causal, stationary and partially adaptive (not all weights are learned) transductions in which the encoding function is realized by a fixed recurrent layer (the reservoir) and the output function is realized by a linear output layer (the readout). The key idea of RC paradigm is to exploit simple properties of dynamical systems allowing the reservoir to discriminate among different input histories even without the training of recurrent weights [52]. Thereby, only the non-recurrent part (the readout) is trained. Overall, the RC framework provides theoretical and empirical background to analyze recurrent architectures and at the same time to design efficient RNN models. The RC paradigm includes several RNNs classes, such as Echo State Networks (ESNs) [98, 99, 100], Prediction Fractal Machines (PFMs) [190, 191], Liquid State Machines (LSMs) [125], BackPropagation Decorrelation [177, 179] and Evolino [168]. In this thesis, we focus on the ESN model.

Echo State Networks

The ESN model is an efficient implementation of the RNN approach within the RC framework. The architecture is composed by two parts, a non-linear recurrent layer called reservoir and a linear output layer (the readout). The reservoir is randomly initialized and left untrained, while, the readout computes the output of the network exploiting the temporal state representation developed by the reservoir part. Figure 2.7 shows the ESN architecture.

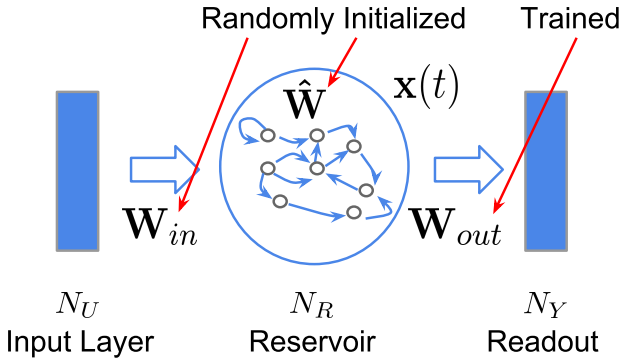


Figure 2.7: The ESN architecture.

In this thesis, we consider a variant of ESN called Leaky Integrator ESN (LI-ESN) [100] in which the resevoir contains leaky integrator units. Omitting the bias terms in the following formulas for the ease of notation, the state transition function of LI-ESN is defined as follows:

$$\mathbf{x}(t) = \tau(\mathbf{u}(t), \mathbf{x}(t-1)) = (1-a)\mathbf{x}(t-1) + a\,\mathbf{tanh}(\mathbf{W}_{\text{in}}\mathbf{u}(t) + \hat{\mathbf{W}}\mathbf{x}(t-1)), \quad (2.21)$$

where $\mathbf{u}(t) \in \mathbb{R}^{N_U}$ and $\mathbf{x}(t) \in \mathbb{R}^{N_R}$ are respectively the input and the reservoir state at time t, $\mathbf{W}_{\text{in}} \in \mathbb{R}^{N_R \times N_U}$ is the matrix of the input weights, $\hat{\mathbf{W}} \in \mathbb{R}^{N_R \times N_R}$ is the matrix of the recurrent weights, $a \in [0, 1]$ is the leaky parameter and **tanh** represents the element-wise application of the hyperbolic tangent activation function. In the following, we also use the term *shallowESN* to refer to LI-ESN model.

Echo State Property

The reservoir parameters are initialized to satisfy the Echo State Property (ESP) [99, 213, 52]. This initialization determines asymptotic properties in the state

dynamics encoded from input sequences. It is defined as follows:

$$\begin{aligned} \forall \mathbf{s}_n(\mathbf{u}) = [\mathbf{u}(1), ..., \mathbf{u}(n)] \in (\mathbb{R}^{N_U})^n, \forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^{N_R} : \\ \|\hat{\tau}(\mathbf{s}_n(\mathbf{u}), \mathbf{x}) - \hat{\tau}(\mathbf{s}_n(\mathbf{u}), \mathbf{x}')\| \to 0 \text{ with } n \to \infty, \end{aligned} \tag{2.22}$$

where $\mathbf{s}_n(\mathbf{u}) = [\mathbf{u}(1), ..., \mathbf{u}(n)]$ is an input sequence of length n and $\hat{\tau}$ is the global state function defined in 2.7. Equation 2.22 means that at the increasing of the sequence length, the norm of the difference between the two states, i.e. their distance, encoded starting from two different initial states tends to 0. Thereby, the state dynamics progressively lose the information regarding the initial states of the network focusing the encoding process on the input information. The ESP is also related to the contractivity of the state dynamics that allows the ESN to be an universal approximator of finite memory machines [192, 52]. Moreover, the contractivity bounds the dynamics of the ESN in a region of the state space characterized by *Markovian* properties providing an intrinsic suffix-based clustering of sequences [192, 51, 52]. A necessary condition for the asymptotic stability of the null state with zero input and a sufficient condition for the ESP are presented in [98]. The former condition consists in setting the spectral radius (i.e., the maximum absolute eigenvalue) of $\hat{\mathbf{W}}$ less than 1:

$$\rho(\hat{\mathbf{W}}) < 1; \tag{2.23}$$

while the sufficient condition for ESP states that the largest singular value of $\hat{\mathbf{W}}$ must be less than 1:

$$\sigma(\hat{\mathbf{W}}) < 1. \tag{2.24}$$

The largest singular value is the euclidean norm of $\hat{\mathbf{W}}$ then the sufficient condition 2.24 is equivalent to $\|\hat{\mathbf{W}}\|_2 < 1$. In practical applications the condition on the spectral radius 2.23 is preferred, since the sufficient condition is considered too restrictive [98]. A more recent work provides a less restrictive sufficient condition for ESP in terms of diagonal Schur stability [213].

In the case of LI-ESN the values in matrix $\hat{\mathbf{W}}$ are randomly selected from a uniform distribution over $[-1, 1]$, and then rescaled to satisfy the following condition:

$$\rho\left((1-a)\mathbf{I} + a\hat{\mathbf{W}}\right) < 1. \tag{2.25}$$

The values in matrix $\mathbf{W}_{\text{in}}$ are randomly selected from a uniform distribution over $[-scale_{\text{in}}, scale_{\text{in}}]$, where $scale_{\text{in}}$ is the input scaling parameter.

In experimental assessments, for each reservoir hyper-parameterization a number of different (independently initialized) instances are considered, all with the same reservoir hyper parameters, but generated using different random seeds. In the following, we refer to such instances as reservoir guesses.

Output Computation and Training of ESNs

The output of the network at time t is computed through a linear combination between the readout and reservoir states, as follows:

$$\mathbf{y}(t) = \mathbf{W}_{\text{out}}\mathbf{x}(t). \tag{2.26}$$

Readout training consists in finding the values of $\mathbf{W}_{\text{out}}$ such that the norm $\|\mathbf{W}_{\text{out}}\mathbf{X} - \mathbf{Y}_{\text{target}}\|_2^2$ is minimized, which is equivalent to solving the linear system:

$$\mathbf{W}_{\text{out}}\mathbf{X} = \mathbf{Y}_{\text{target}}, \tag{2.27}$$

where $\mathbf{X} = [\mathbf{x}(1), ..., \mathbf{x}(N_T)] \in \mathbb{R}^{N_R \times N_T}$ is computed by state trasition function 2.21, $\mathbf{Y}_{\text{target}} = [\mathbf{y}(1), ..., \mathbf{y}(N_T)] \in \mathbb{R}^{N_Y \times N_T}$ is the matrix of target values of the *training set* and N_T is the number of samples of the training set. Typically, the training is performed using pseudo-inversion or ridge regression methods [121] computed through singular value decomposition (SVD) and normal equations approaches. In the case of pseudo-inversion through normal equation the output weight matrix $\mathbf{W}_{\text{out}}$ is computed in the following way:

$$\mathbf{W}_{\text{out}} = \mathbf{Y}_{\text{target}}\mathbf{X}^\intercal(\mathbf{X}\mathbf{X}^\intercal + \lambda_r\mathbf{I})^{-1}, \tag{2.28}$$

where $\lambda_r \geq 0 \in \mathbb{R}$ is the Tikhonov regularization term.

Computational Cost of ESNs

Since in ESN architectures only the output layer is trained, they are considered very efficient models. The cost of computing the state matrix $\mathbf{X}$ from an input sequence of length N_T is the following:

$$\mathcal{C}_{\text{encoding}} = \mathcal{O}(N_R^2 N_T). \tag{2.29}$$

Moreover, typical pseudo-inverse approaches used for the training of readout require:

$$\mathcal{C}_{\text{pinv}} = \mathcal{O}(min(N_R N_T^2, N_R^2 N_T)). \tag{2.30}$$

Therefore, considering the case in which $N_R < N_T$, the cost of training can be rewritten as follows:

$$\mathcal{C}_{\text{readout}} = \mathcal{O}(N_R^2 N_T). \tag{2.31}$$

Overall, the cost of training ESNs is:

$$\mathcal{C}_{\text{esn}} = \mathcal{C}_{\text{encoding}} + \mathcal{C}_{\text{readout}} = \mathcal{O}(N_R^2 N_T). \tag{2.32}$$

For the learning of fully trained RNNs the network's state is calculated for each number of epochs N_E. Moreover, in this case, the learning algorithm computes also the delta rule for each time-step of the input sequence. Then, the total cost results as in the following:

$$\mathcal{C}_{\text{fullyRNN}} = N_E N_D \, \mathcal{C}_{\text{encoding}} = \mathcal{O}(N_R^2 N_T N_E N_D), \tag{2.33}$$

where N_D is the cost of delta rule computation.

Overall comparing Equation 2.33 with Equation 2.32 we can note that ESNs are extremely more efficient than fully training RNNs, since in learning algorithms of typical RNNs the cost of the state computation in Equation 2.29 is also multiplied by the cost of the delta rule and by the number of epochs (typically in the order of thousands).

Intrinsic Plasticity Pre training in ESNs

In the context of RC, a well known unsupervised approach for reservoir adaptation is called Intrinsic Plasticity (IP) [171, 178, 195, 121]. This approach is based on maximizing the entropy of the output distribution of the reservoir units. Specifically, IP aims at adjusting the parameters of the activation function (e.g., gain and bias values) in a way that the activation function distribution fits the maximum entropy distribution of the reservoir units. As explained in [171], since reservoir units have a distribution with support $[-\infty, +\infty]$, the distribution with maximum entropy is the Gaussian. The adaptation of the parameters is obtained through an iterative algorithm that minimizes the Kullback-Leibler divergence between the activation function distribution and the target distribution. In our context, the application of

the reservoir activation function for a generic reservoir unit v can be expressed as $f(v) = \mathbf{tanh}(\mathbf{g}\, v + \mathbf{b})$, where $\mathbf{g}$ and $\mathbf{b}$ are respectively gain and bias vectors of the activation function f. Therefore, Equation 2.21 is rewritten as:

$$\mathbf{x}(t) = \tau(\mathbf{u}(t), \mathbf{x}(t-1)) = (1-a)\mathbf{x}(t-1) + a\,\mathbf{tanh}(\mathbf{g}\,\mathbf{W}_{\text{in}}\mathbf{u}(t) + \hat{\mathbf{W}}\mathbf{x}(t-1) + \mathbf{b}). \tag{2.34}$$

The IP training procedure is performed on each reservoir unit on the basis of the IP rule [171] defined as:

$$\begin{aligned} \Delta b &= -\eta_{\text{IP}}(-(\mu_{\text{IP}}/\sigma^2_{\text{IP}}) + (f(v)/\sigma^2_{\text{IP}})(2\sigma^2_{\text{IP}} + 1 - f(v)^2 + \mu_{\text{IP}} f(v))), \\ \Delta\mathbf{g} &= \eta_{\text{IP}}/\mathbf{g} + \Delta\mathbf{b}\, v, \end{aligned} \tag{2.35}$$

where μ_{IP} and σ_{IP} are the mean and the standard deviation of the Gaussian distribution (that is the target of the adaptation process), η_{IP} is the learning rate, $\Delta\mathbf{g}$ and $\Delta\mathbf{b}$ respectively denote the update values for gain and bias of the IP iterative learning algorithm.

Applications of ESNs

The success of ESNs obtained in artificial tasks [98, 99, 200, 52] characterized by chaotic time-series prediction and signal classification has contributed to stimulate the interest of neural networks' community.

For what regards real-world applications, ESNs obtained good results on tasks characterized by noisy, continuous and heterogeneous signals. Examples of these kind of tasks are represented by Human Activity Recognition [141, 131, 140] and Health Informatics [65, 64, 149, 113].

A relevant application field for ESN models concerns time-series prediction. Applications for the prediction of Market prices are presented in [119, 38, 35]. Forecasting systems for the prediction of person movements are introduced in [81, 7]. Other works explore spatio-temporal forecasting for meteorological prediction [128, 210].

Moreover, the ESN models are widely used to develop different Control System applications [28] such as, robotic arm control [197], pneumatic muscle control [208], oil well control [103] and motor control [160].

The ESN model is also implemented in many robotic applications such as robot modelling [148], navigation [78], localization [40] and soft robotic arm controller [112].

The applications described above are characterized by small/medium datasets. Some works attempted to address applications characterized by a medium/big quantity of data. Relevant results are obtained by ESN architectures in continuous speech recognition task [194, 193]. Other works propose the use of RC approaches in computer vision tasks [207, 101, 164]. A difficulty of ESN architectures on tasks characterized by medium/big datasets is that they needs a bigger number of recurrent units than other RNN approaches. In particular, in large reservoir the issues are represented by the computational time needed for the initialization procedure and by the quantity of memory required for the state computation of the network.

2.4.2 Architectural Design in RC paradigm.

Depending on the task, architectural factors in RC paradigm are crucial to achieve rich dynamics of the global transition function corresponding to good predictive performance [52]. Most works are focused on the study of the recurrent part of RC models. In particular, different configurations of recurrent connections can affect the memory capacity of the model and the ability to represent temporal inputs characterized by multiple time-scales dynamics. The study of recurrent connections is typically performed analyzing architectural topologies and considering different algebraic characterizations of recurrent matrices.

Studies presented in [206, 211] propose solutions to overcome difficulties encountered by standard ESNs in representing signals characterized by multiple time-scales dynamics. Some aspects of these studies are related to the kind of interconnection between neurons. In order to decouple the dynamics of the neurons, and then to enhance the multi-scales processing, some works explored neuroevolution approaches that coevolves separate sub-populations of neurons [206, 168]. Other methods proposed ESN architectures composed by many decoupled sub-reservoirs in which every sub-reservoir can have its own state dynamic [211, 151, 17]. Recently, hierarchical composition of ad hoc modules of ESNs shows promising results in time-series tasks [95, 194, 193]. In particular, in [95] the multi-scales processing is obtained through a gradient descent approach applied to a hierarchy of ESNs. Whereas, in [194, 193] a hierarchy of ESNs is progressively trained through direct methods in which each level is individually trained on the output data of previous level.

Other studies analyze how the properties of recurrent matrices can determine

the performance of the network. Indeed, the structure of recurrent matrices affects the way in which neurons are connected determining the dynamics involving in the reservoir. An approach consists in the study of orthogonal recurrent matrices [204, 76]. A simple strategy proposed in [76] is generating reservoir connections through random permutation matrices. The benefit achieved by permutation matrices is also studied in terms of memory capacity and predictive capability [20]. In general, the use of orthogonalization approaches for recurrent matrices considerably increase the memory capacity of the network [44, 45].

Another approach consists in introducing a notion of time-scale in the dynamics of the network state by using integrator units [100, 88]. In particular, the parameter of leaky integrator units allows the reservoir to control the speed of the state's dynamic. In general, recurrent layers can be studied from the filtering point of view [120, 89, 209, 170]. In particular, in [209] it is proposed a band-pass filter integrator designed to handle specific frequency bands while in [170] the dynamic of the reservoir's state is controlled by three integrators applied in three different ways, before, over and after the activation function.

A recent aspect studied in literature regards the design of architectures characterized by low complexity [154, 155]. The aim is to find simple reservoir topologies able to simplify the encoding process and to obtain good performance. These works show that good results can be achieved with simple architectures such as linear, circular, delay lines and deterministic reservoirs [27, 154, 155, 181, 4].

2.5 Deep Learning Paradigm

Deep Learning (DL) [72, 116, 165] is a ML paradigm that deals with the study of NNs characterized by layered architectures. These architectures are composed by multiple processing layers with linear and non-linear transformations. Each layer can be trained in supervised or unsupervised way. In the last years, DL approaches [72] achieved impressive results in many fields such as speech recognition [73, 134, 193], machine translation [185], image classification [110] and mastering the game of Go [187, 175]. The characteristic of the DL paradigm is to provide compositionality of internal representations in the architecture. Higher layers features are derived from lower layers features in order to form a hierarchical structure in which each level represents a different data abstraction. Furthermore, a stack of layers enables to make a complex representation based on a hierarchy of simpler concepts. The abstraction of data representations of a given layer increases with the increasing of

the distance between that layer and the input layer.

A major issue addressed by DL paradigm is the supervised training of layered neural networks. It is well known that training deep models by using gradient descent approaches is difficult [72, 85, 14]. In particular, gradient descent approaches tend to fall in local minimum solutions contributing to increase the computational cost of the training procedure [70]. Moreover, gradient descent approaches can suffer from vanishing and exploding gradients [15]. Relevant solutions to address the difficulty of training NNs are proposed in [15, 70, 146]. In this regard, most works propose learning approaches such as *gradient norm clipping* and *soft constraints* [146] or architectural solutions [88, 31, 80].

Other solutions to address the difficulty of training are explored in studies such as, approximation algorithms for training [84], *pre training* [43], *rectified units* [71], *dropout* [86] and SGD [158, 22, 21, 107]. A major solution that contributed to achieve an impressive speedup of training efficiency consists in *Graphics Processing Unit* (GPU) parallelization [218, 3].

Overall, the use of solutions proposed in these studies, the hardware improvements and the possibility to easy parallelize the computation of NNs models through GPUs allowed DL models to obtain impressive results in real-world applications and to progressively popularize DL paradigm outside the Machine Learning communities.

2.5.1 Deep Neural Networks

In this thesis we deal with *deep* NNs. A deep NN architecture is a deep model characterized by a stack of layers composed by (typically non-linear) neurons [73, 134, 185, 110, 72].

An interesting debate in DL community concerns the characterization between *deep* and *shallow* (i.e., non-deep) models. Figure 2.8 shows the comparison between a *deep* and a *shallow* architecture. While shallow NNs typically refers to those models composed by a single hidden layer, deep NNs are characterized by a stack of non-linear transformations represented by a hierarchy of multiple hidden layers. Both kinds of models are universal approximators of continuous functions. However, empirical and theoretical results on specific scenarios [118, 124, 136] suggest that deep models have better abilities than shallow NNs in providing a distributed, hierarchical and compositional representation of the input information exploited in application tasks to improve performance.

A major class of deep NNs is composed by *Convolutional Neural Networks* (CNNs)

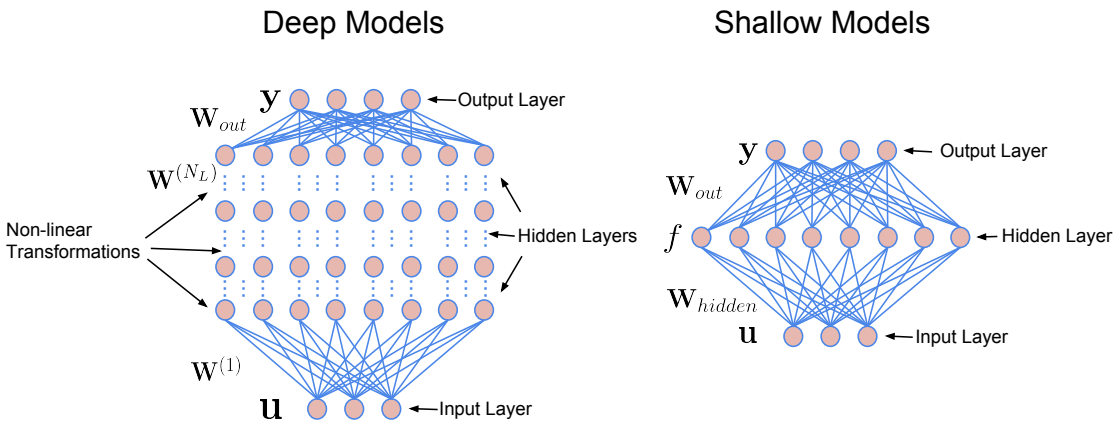


Figure 2.8: Comparison between deep and shallow models.

[36, 115, 117]. They are a class of hierarchical NNs characterized by *convolutional layers* able to process 2-dimensional inputs suitable for computer vision. In recent works, the combined use of CNNs with the DL paradigm has obtained results that improve the state of the art in application areas such as image recognition and text classification [110, 187, 33, 80], computer vision [105] and automatic playing [133, 174, 175]. The weights of the convolutional layers are composed by a set of trainable *filters*. Each filter of a convolutional layer is connected with a small *receptive field* (a sort of input window) and it is applied through all input sequence. The computation of a filter consists in a convolution operation and forms a *feature map*. The value of the feature map $\mathbf{F}(i,j)$ is computed as follows:

$$\mathbf{F}(i,j) = \sum_t \sum_s \mathbf{I}(i+t, j+s)\mathbf{W}(t,s), \tag{2.36}$$

where $\mathbf{I} \in \mathbb{R}^{n \times h}$ is a 2-dimensional input (e.g., an image) and $\mathbf{W} \in \mathbb{R}^{t \times s}$ is a filter connected to a receptive field of width t and height s. The units of a feature map share the same weights. This approach, called *weight sharing*, allows the features to be detected independently from their position in the input, thus providing the property of translation invariance [115].

Another interesting aspect of CNNs is the use of pooling layers after the convolutional layers. A pooling layer subdivides a feature map in many non-overlapping areas and for each area it computes an aggregation function such as maximum or average pool. This regularization method reduces the dimensionality of data representations and consequently causes a reduction of the model complexity allowing the NN architecture to address overfitting issues.

The CNNs are trained using the typical learning approaches of MLPs such as BP and SGD [158, 115, 117, 22].

2.6 Deep Recurrent Neural Networks

Here, we present an overview of literature approaches for deep RNNs. In Section 2.6.1, we introduce deep RNNs characterized by hierarchies of recurrent layers. In Section 2.6.2, we present a definition of depth in deep recurrent architecture. In Section 2.6.3, we present ad-hoc approaches to force multiple time-scales dynamics among the layers. In Section 2.6.4, we describe deep transition approaches to add depth in recurrent architectures. Finally in Section 2.6.5, we describe early works based on hierarchies of ESN modules.

2.6.1 Hierarchies of Recurrent Layers

Deep RNNs are typically characterized by hierarchies of recurrent layers [167, 83, 82] in which lower layers are connected to higher layers. Figure 2.9 shows the architecture of a deep RNNs composed by a stack of recurrent layers. In general, the

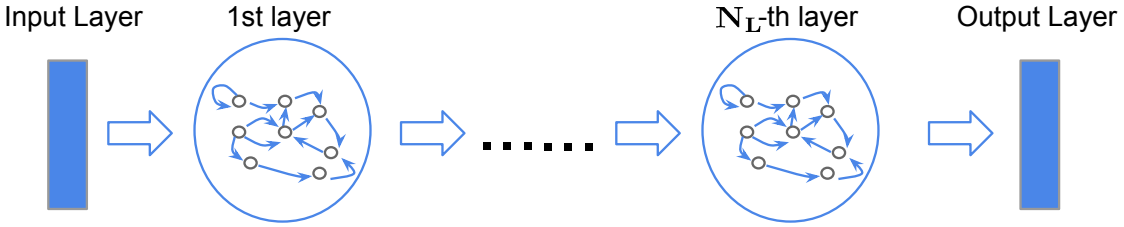


Figure 2.9: A deep RNN architecture with multiple recurrent layers.

use of a stack of recurrent layers enables to operate at different time scales [167, 83, 95, 82, 193, 30, 3]. This characteristic allows the deep RNN to learn a rich temporal representation from input signals characterized by multiple time-scales dynamics. Indeed, empirical experiments showed that deep ESNs outperform shallow RNNs on real-world applications [82, 73] characterized by multiple time-scales dynamics. In particular, the studies conducted on language modeling tasks in [82] show that stacked RNNs can learn a hierarchy of time-scales dynamics over the layers. Thereby, higher recurrent layers have progressively long memory of input perturbations.

2.6.2 Depth of Deep Recurrent Architectures

As described in Section 2.5.1, empirical and theoretical results show that *depth* in deep NNs leads to more expressive models [136]. However, while in feed-forward NNs the depth is given by the number of non-linear hidden layers, in the case of recurrent architectures the definition of depth is not trivial [144].

The feedback connections implemented in a recurrent layer causes a loop over time in the computation of the network's state. A way to represent this computation is to consider an unfolded recurrent architecture over the time-steps of an input sequence. The Figure 2.10 shows the unfolded structures of a 1-layered (i.e., an SRN) and a 3-layered RNN (i.e., a deep RNN with a stack of 3 recurrent layers) over time. The resulting unfolded architectures are feed-forward NNs which are "deep"

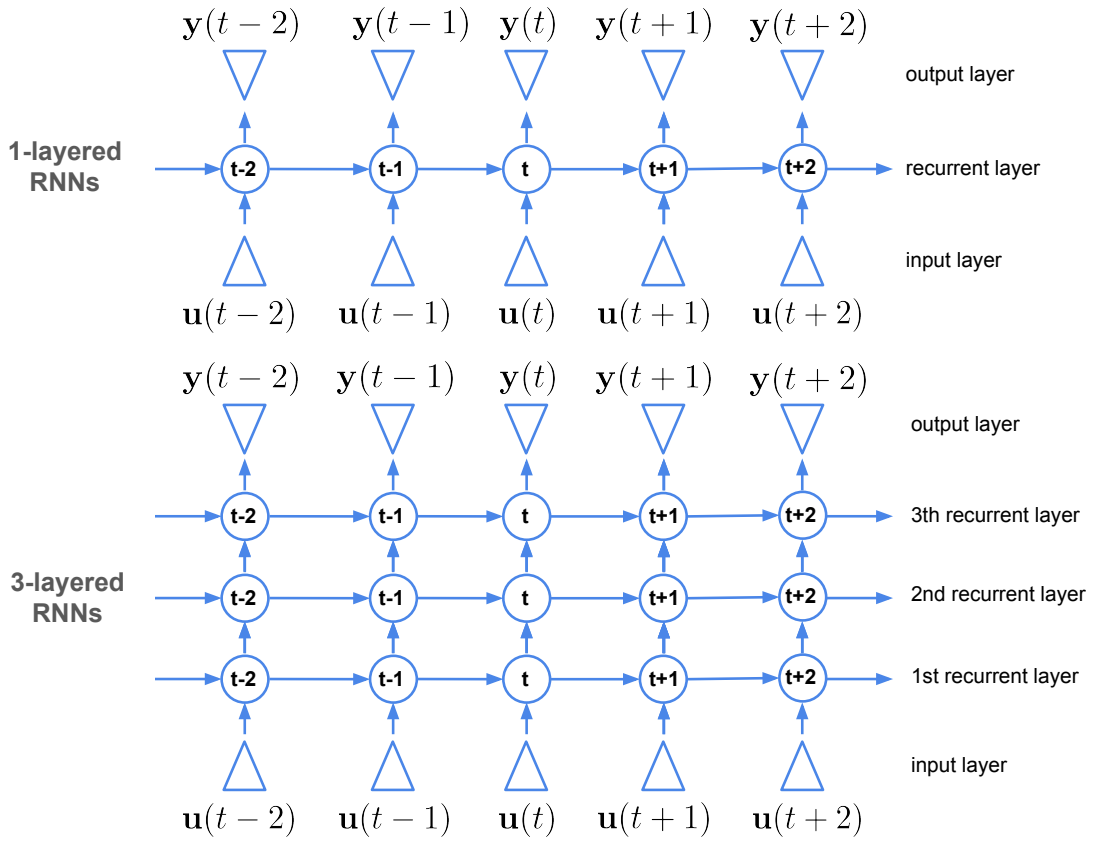


Figure 2.10: Comparison between unfolded shallow and deep RNNs. Circles represent unfolded recurrent layers and triangles represent inputs and outputs over time.

over time. However, if we consider a single time-step t we can note that, in the 1-layered architecture, the computation flows through a single hidden layer between input and output layers. While in the 3-layered architecture the computation

flows through 3 hidden layers. Therefore, the 1-layered and 3-layered RNNs have respectively $depth = 1$ and $depth = 3$. Typically, an RNN with $depth = 1$ is considered "shallow", while, a recurrent architecture with $depth > 1$ is considered "deep". Therefore, 1-layered and 3-layered RNNs are "shallow" and "deep" [121, 144, 82, 219], respectively.

A detailed study on complexity measures regarding depth in deep recurrent architectures is presented in [219].

2.6.3 Forcing Multiple Time scales Over Layers

Some works in literature aimed at obtaining multiple time-scales dynamics in layered RNNs by forcing the process at different frequencies over the layers [83, 30, 193].

In particular, [83] introduced *hierarchical RNNs* (HRNNs) characterized by hierarchies of recurrent layers with different time-delays in which each of them "works" at different time scales. Figure 2.11 shows an example of an unfolded HRNN. In this example, the first recurrent layer processes the input at each time-steps.

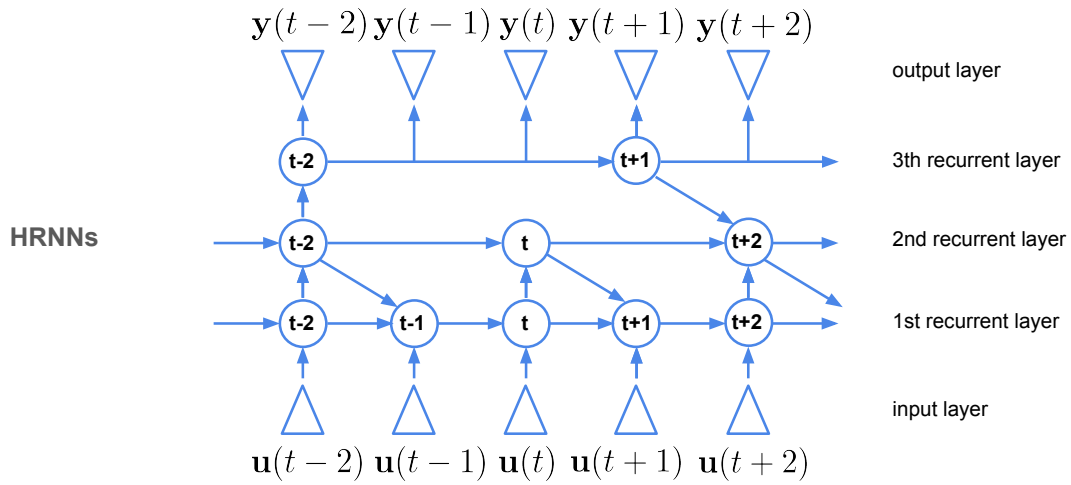


Figure 2.11: Example of an HRNN architecture. In particular, in this example, the first layer implements a 1-step time-delay, the second layer implements a 2-step time-delay and the third layer implements a 3-step time-delay. Circles represent unfolded recurrent layers and triangles represent inputs and outputs over time.

Moreover, the second recurrent layer, that is fed by the output of the first layer, processes the data every 2 time-steps. Finally, the third layer processes the data every 3 time-steps. In this way, we impose a progressively slower speed of layer

dynamics at higher levels in the architecture. This allows the architecture to develop a hierarchy of temporal features.

Empirical experiments on controlled scenarios show that HRNN outperforms shallow RNN in memory tasks. Moreover, it is shown that HRNN performs progressively better adding more recurrent layers with different time-delays.

A model based on the use of a hierarchy of RNN modules that operate at different velocities of processing is presented in [193] (see Section 2.6.5 for details regarding this architecture).

2.6.4 Deep Transitions

A further study on architectural design aspects proposes the use of deep transitions among architectural parts [144]. Differently to stack multiple recurrent layers, this approach consists in adding multiple feed-forward layers in recurrent and output parts of the network. Figure 2.12 shows an example of recurrent architectures with deep transitions so called Deep Output Transition RNN (DOT-RNN) [144]. In

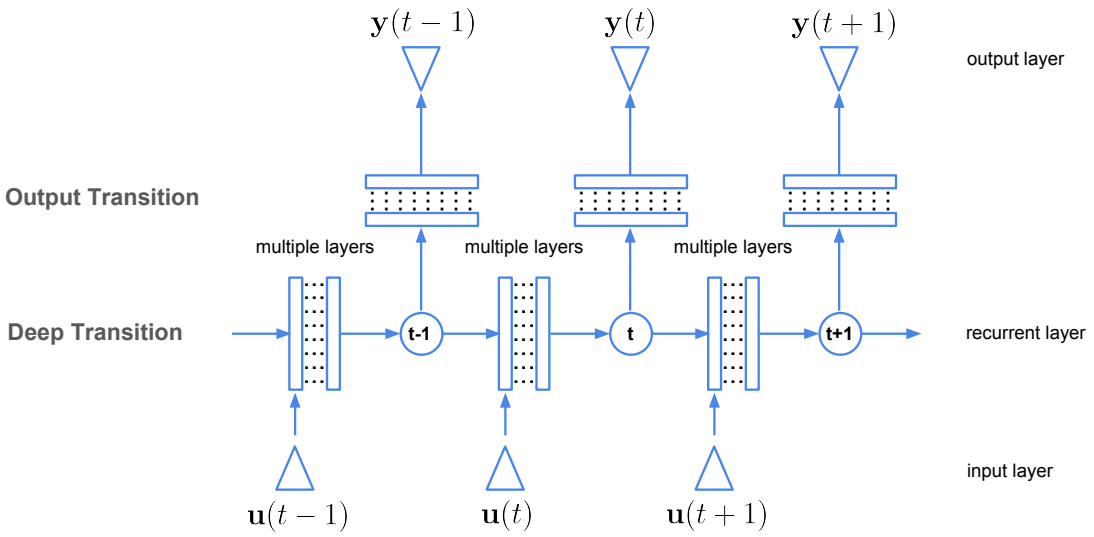


Figure 2.12: An example of an unfolded recurrent architecture that implements deep transitions in recurrent and output layers so called DOT-RNNs [144]. Circles represent unfolded recurrent layers, triangles represent inputs and outputs over time and rectangles are multiple non-linear layers.

particular, DOT-RNN implements two kind of non-linear transformation called Deep Transitions and Output Transitions. A Deep Transition is an MLP that implements the state transition function of the network, while, an Output Transition is an MLP that implements the output computation of the network. Interestingly, we can note from Figure 2.12 that for both Transitions we obtain an architecture with

a *depth* > 1 if we consider MLPs with at least 2 hidden layers. The experimental results highlighted that DOT-RNN outperforms shallow RNN on real-world tasks [144].

2.6.5 Early works on modular ESNs

Early works based on hierarchical networks are mainly focused on *ad hoc* modular organizations of Echo State Networks [121, 95, 194, 193, 101]. In particular, [95] introduces a stack of ESNs in which each ESN constitutes a layer of the hierarchy. The higher layers generate an output that serves as a voting vector for the lower ones. Each layer of the hierarchical architecture is trained by a gradient descent approach. The downside of such model is that, as fully trained RNNs, it suffers from a similar problem of vanishing gradients. Another work proposes the use of a cascade of ESNs each individually trained on the output of the previous one [193]. Thereby, the higher layers can correct the errors made by the preceding layers. Figure 2.13 shows an example of a cascade of ESNs. Moreover, the approach considers different RC parameters in the higher modules in order to impose different velocities of processing among the hierarchy. Such model obtains promising results

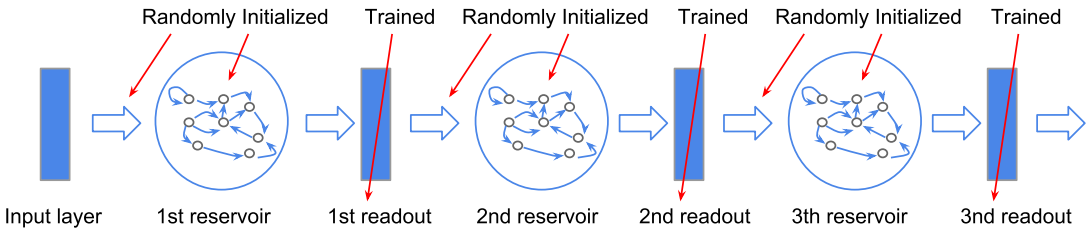


Figure 2.13: An example of a cascade of ESNs.

in acoustic modeling on a challenging continuous speech recognition task so called TIMIT [194, 193].

Although these models propose hierarchical RC architectures, the developing of temporal features characterized by different multiple time-scales dynamics is based on the learning of all layers of the hierarchy. Contrariwise, in this thesis (see Chapter 3) we introduce a novel kind of architecture composed by a hierarchy of recurrent layers randomly initialized and left untrained able to intrinsically develop hierarchical temporal features even without the learning of recurrent weights.

2.6.6 Open Issues

The design of models according to DL paradigm enables to exploit the expressiveness provided by a stack of non-linear transformations [136]. Moreover, experimental analysis show that deep models provide a distributed, hierarchical and compositional representation of the input information [216]. In particular, [216] presents a method to analyze the features developed by a CNN starting from an image. This analysis shows that first layers direct their attention towards specific details such as borders and shapes. Instead, central layers are focused on patterns such as eyes and noses and successive higher layers are focused on more abstract concepts such as faces or poses of objects. The experiments conducted in [216] highlighted that these representations affect the performance achieved by the model.

In this thesis, we deal with the analysis and the design of deep recurrent architectures suitable to process time series characterized by multiple time-scales dynamics. Although results reported in literature works reported that deep RNNs obtain better results than shallow RNNs (as described in Section 2.6), there are still fundamental issues to be investigated [3, 72]. Experimental analysis shows that deep RNNs create different time-scales dynamics at different layers [82, 144], and therefore a hierarchical representation of the temporal information. However, these results are typically obtained through the learning of all recurrent layers or with the use of different time-delays. Therefore, the study of the merits of layering and its inherent impact on temporal features provided by the network deserves to be investigated. Another related aspect to be investigated is the impact of temporal representation developed by layering on the performance achieved by the model in tasks characterized by multiple time-scales dynamics.

Other open issues regard the training of deep recurrent architecture. Indeed, the application of gradient descent algorithms to deep and recurrent architectures are well-known issues [72, 85, 14, 146]. Typically, deep RNNs implement few layers ($<$ 10) in state-of-the-art applications. Moreover, the training of all recurrent layers is very difficult.

For what regards the design of deep RNNs, the typical approach is to perform an expensive trial and error approach based on model performance. In order to determine the number of layer of the deep architecture, we need to evaluate all possible configurations.

In this thesis, we aim to address these issues by resorting to RC paradigm (described in Section 2.4.1) that represents an extremely efficient solution for RNN

modelling. Contrariwise to modular RC architectures described in Section 2.6.5, we propose a novel class of deep RNN models (that we introduce in the next Chapter) based on a stack of randomly initialized and untrained recurrent layers. This allows us to study the effect of different parameters of RC models on the dynamical behavior of deep RNNs and to investigate the architectural factors of deep recurrent models in a decoupled fashion with respect to the learning aspects of the dynamical part of the networks. Finally, we aim to define quantitatively and qualitatively measures based on frequency analysis for the design of deep RNNs (see Chapter 4).

Deep Reservoir Computing

The content presented in this Chapter is based on works published in [61, 63, 54] (see Section 1.5 for details).

In this Chapter we introduce Deep Reservoir Computing as a novel class of randomized deep RNNs. In Section 3.1, we describe the aspects that motivate the developing and the analysis of deep RC architectures. Moreover, in Section 3.2 we introduce the Deep Echo State Network (DeepESN) model characterized by a hierarchy of randomized recurrent layers. In Section 3.3, we study and analyze the intrinsic temporal characterizations of deep RNNs by using DeepESN models. In Section 3.4, we introduce the Linear DeepESN (L-DeepESN) to study the intrinsic role of layering by means of frequency analysis. In Section 3.5, we discuss the outcomes of this Chapter, and then, we describe related works subsequently published in this area. Finally, conclusions are drawn in Section 3.6.

3.1 Introduction

Recent studies on deep RNNs have stimulated a growing interest under both theoretical and applicative points of view [219, 166, 144, 82], especially in regard to the possibility of developing a hierarchical processing of temporal data. Indeed, the ability to represent dynamical features at multiple levels of abstraction allows to capture more naturally the temporal structure of the data whenever it is intrinsically characterized by a multiple time-scales organization. Among the others, language [144], speech [73] and text processing [82] represent notable examples of application areas involving time-series data with this type of characterization. Besides, the capability of modeling multiple time scales in recurrent networks dynamics has proved effective also as a mean to deal with long-term dependencies, as evidenced e.g. in [137, 83] and, more recently, in [72, 146].

Some works in literature aimed at obtaining multiple time-scales dynamics in a layered RNN architecture. One approach consists in progressively subsampling the input to the higher layers [83], forcing the different layers to operate at different frequencies [193]. Another approach consists in learning all the weights in the stack of recurrent layers, which is a difficult and extremely time consuming process even using GPUs and can require ad hoc incremental training strategies in order to be effective [82].

As described in Section 2.6.6, the state of the art in this respect is still in its infancy with many open challenges [3, 72], and some intuitions present in literature deserve further research and critical assessments. In particular, the observation that stacking layers of recurrent units inherently creates different time-scales dynamics at different layers [82, 144], and therefore a hierarchical representation of the temporal information *per se*, deserves to be investigated and analyzed.

A starting point for our analysis in this regard is represented by the observation that stacking recurrent layers can be actually interpreted as the application of a set of constraints to the architecture of a fully connected RNN (with the same number of units). Such constraints involve the pattern of connectivity among the recurrent units (i.e. avoiding connections from higher layers to lower layers), which affects the flow of information and the dynamics of sub-parts of the network state. Moreover, the architectural restrictions also concern the connectivity with the input layer (i.e. allowing only to the units in the first layer to be fed directly by the input), influencing the way in which the external input information is seen by recurrent units progressively more distant from the input layer (such architectural aspects are discussed in Section 3.2.1 and illustrated in Figure 3.3).

This motivates us to a critical assessment of the possible and effective merits of a layered structure for recurrent architectures and to propose different approaches to achieve a hierarchy in temporal representation by efficient deep recurrent models.

To this aim, the modeling proposal is based on RC paradigm (described in Section 2.4.1), which represents a state-of-the-art approach for extremely efficient RNN modeling. Moreover, and more importantly for the analysis purposes, the RC approach yields the possibility to investigate the architectural factors of deep models in a decoupled fashion with respect to the learning aspects of the dynamical part of the networks. This type of analysis on the one hand can provide insights on the true merits of learning of deep RNN dynamics, and on the other hand it allows to propose efficiently trained models for multiple time-scales processing of temporal data.

Previous works on hierarchical organizations of RC networks mainly focused on *ad hoc* modular organizations of Echo State Networks [121, 95, 193], but still lack of a general view over the effective potentiality and emerging properties of deep architectures of layered reservoirs. In particular, since different parameters of RC models strongly affect their dynamical behavior and performance, their relationships with layering deserve a systematic investigation, still missing in literature. Such investigation allows us to study proposals on what aspects ruling the dynamics of reservoir models can amplify the potential benefits of a deep architecture (and vice versa), in particular for the timescale dynamics differentiation, as well as for the effect of known RC techniques, such as IP pre training (described in Section 2.4.1), to enhance the richness of state dynamics (measured as the entropy of reservoir states).

Overall, in this Chapter we show how to obtain, enhance, and quantify the occurrence of different time scales (or amplify the richness of the state dynamics) in deep recurrent architectures in order to address the main open issues described in Section 2.6.6 regarding the deep RNN field: (i) Why introduce layering into recurrent architectures? (ii) What is the intrinsic (independent from learning) architectural effect of layering on the hierarchical temporal dynamics developed by a deep RNN? (iii) Is it possible to keep the advantage of deep learning for RNN (e.g. in terms of multiple time-scales representation of temporal data) by using an efficient approach such is RC? (iv) What is the role of the hyper parameters that rule RC network dynamics within a layered organization of the reservoir?

3.2 Deep Echo State Networks

Here we introduce the main model that we take into consideration, called *Deep Echo State Network* (DeepESN).

Architecture

The architecture of a DeepESN is characterized by a stacked hierarchy of reservoirs, as shown in Figure 3.1. At each time-step t, the first recurrent layer is fed by the external input $\mathbf{u}(t)$, while each successive layer is fed by the output of the previous one in the stack. Although the pipelined architectural organization of the reservoir in DeepESN allows a general flexibility in the size of each layer, for the sake of simplicity here we consider a hierarchical reservoir setup with N_L recurrent layers

each of which contains the same number of units N_R. Furthermore, in our notation, we use $\mathbf{x}^{(l)}(t) \in \mathbb{R}^{N_R}$ to denote the state of layer l at time t.

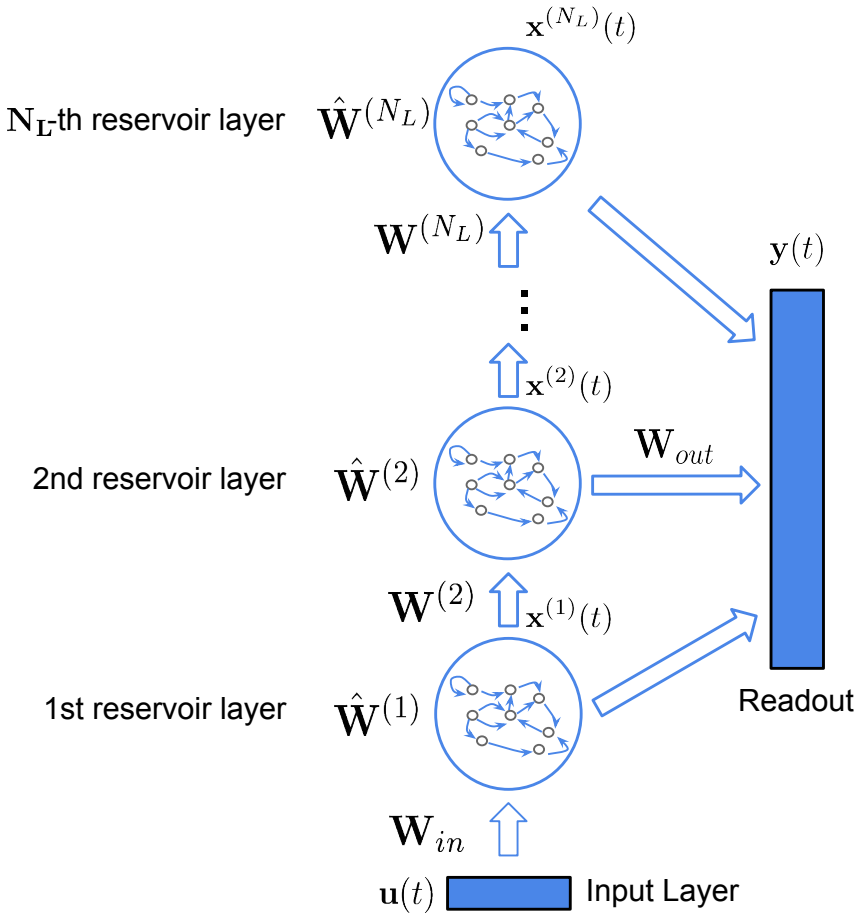


Figure 3.1: Architecture of a DeepESN.

State Computation

Omitting the bias terms for the ease of notation, the state transition function of the first layer is defined as follows:

$$\mathbf{x}^{(1)}(t) = (1 - a^{(1)})\mathbf{x}^{(1)}(t-1) + a^{(1)}\mathbf{f}(\mathbf{W}_{\text{in}}\mathbf{u}(t) + \hat{\mathbf{W}}^{(1)}\mathbf{x}^{(1)}(t-1)), \tag{3.1}$$

while for every layer $l > 1$ it is described by:

$$\mathbf{x}^{(l)}(t) = (1 - a^{(l)})\mathbf{x}^{(l)}(t-1) + a^{(l)}\mathbf{f}(\mathbf{W}^{(l)}\mathbf{x}^{(l-1)}(t) + \hat{\mathbf{W}}^{(l)}\mathbf{x}^{(l)}(t-1)), \tag{3.2}$$

where $\mathbf{W}_{\text{in}} \in \mathbb{R}^{N_R \times N_U}$ is the input weight matrix, $\hat{\mathbf{W}}^{(l)} \in \mathbb{R}^{N_R \times N_R}$ is the matrix of the recurrent weights for layer l, $\mathbf{W}^{(l)} \in \mathbb{R}^{N_R \times N_R}$ is the matrix relative to the

inter-layer connection weights from layer $l-1$ to layer l, $a^{(l)}$ is the leaky parameter at layer l and $\mathbf{f}$ represents the element-wise application of the activation function (in our case the hyperbolic tangent).

Weight Initialization

As in the standard ESN approach, the reservoir component of a DeepESN is left untrained after initialization subject to stability constraints. In the case of DeepESN, such constraints are expressed by the conditions for the ESP of deep RC networks, given in [55]. Accordingly, the weight values in the recurrent matrices $\hat{\mathbf{W}}^{(l)}$, for $l = 1, \ldots, N_L$, are randomly initialized from a uniform distribution, e.g. in $[-1, 1]$, and then are rescaled to satisfy the necessary condition for the ESP of the DeepESN dynamics around the zero state [55]:

$$\max_{1 \leq l \leq N_L} \rho\left((1 - a^{(l)})\mathbf{I} + a^{(l)}\hat{\mathbf{W}}^{(l)}\right) = \max_{1 \leq l \leq N_L} \rho^{(l)} < 1, \tag{3.3}$$

in which we used the notation $\rho^{(l)}$ to denote the effective spectral radius of the reservoir system in the l-th layer. As regards input and inter-layer matrices, the values in $\mathbf{W}_{\text{in}}$ and $\{\mathbf{W}^{(l)}\}_{l=2}^{N_L}$ are randomly initialized from a uniform distribution and then rescaled in the $[-scale_{\text{in}}, scale_{\text{in}}]$ range. Another initialization approach that we consider in the following consists in rescaling $\mathbf{W}_{\text{in}}$ and $\{\mathbf{W}^{(l)}\}_{l=2}^{N_L}$ such that $\|\mathbf{W}_{\text{in}}\| = \sigma$ and $\|\mathbf{W}^{(l)}\| = \hat{\sigma}$, respectively. As in the case of standard RC, in experimental assessments of DeepESNs, for each reservoir hyper-parameterization a number of different (independently initialized) instances are considered, all with the same reservoir hyper parameters, but generated using different random seeds. In the following, we refer to such instances *as reservoir guesses*. The performance of each hyper-parameterization is then obtained by averaging the performance achieved by the corresponding reservoir guesses.

Output Computation

As pertains to the output computation, although different patterns of state-output connectivity have been explored in recent literature in the case of deep recurrent models [82, 144], in this thesis we focus on the case represented in Figure 3.1, in which the states of all the reservoir layers are used to feed the readout. Specifically,

the output of DeepESN network at time t, i.e. $\mathbf{y}(t) \in \mathbb{R}^{N_Y}$, is computed as follows:

$$\begin{aligned} \mathbf{x}(t) &= (\mathbf{x}^{(1)}(t), \dots, \mathbf{x}^{(N_L)}(t)) \in \mathbb{R}^{N_L N_R}, \\ \mathbf{y}(t) &= \mathbf{W}_{\text{out}} \mathbf{x}(t), \end{aligned} \tag{3.4}$$

where $\mathbf{x}(t) = (\mathbf{x}^{(1)}(t), \dots, \mathbf{x}^{(N_L)}(t)) \in \mathbb{R}^{N_L N_R}$ is the global state of the DeepESN as the composition of the states in the different layers and $\mathbf{W}_{\text{out}} \in \mathbb{R}^{N_Y \times N_L N_R}$ denotes the matrix of the output weights. Note that in the case of DeepESN, the readout formulation given in Equation 3.4 expresses a linear combination between the global reservoir state of the network $\mathbf{x}(t)$ and the readout weight matrix $\mathbf{W}_{\text{out}}$, i.e. a weighted sum of the states coming from all the reservoir layers in the architecture. In the training phase, this directly enables the model to differently weight the contribution of the multiple time-scales dynamics developed through the layers of the deep recurrent architecture, thus enhancing the quality of the temporal representation and the ability to approach temporal tasks for which such differentiation in dealing with the different time scales is important.

Training Algorithm

As regards the training algorithm, the output layer is trained by using direct methods as for RC paradigm (see Section 2.4.1). However, in DeepESN case, the state matrix $\mathbf{X}$ is computed by considering the state of all recurrent layers in the hierarchy (as we can note from Equation 3.4). Therefore, the resulting state matrix computed by Equation 3.4 is $\mathbf{X} = [\mathbf{x}(1), ..., \mathbf{x}(N_T)] \in \mathbb{R}^{N_L N_R \times N_T}$ in which the rows represent the neurons' states from all layers and the columns represent time-steps.

Finally, we compute the trained weights of $\mathbf{W}_{\text{out}} \in \mathbb{R}^{N_Y \times N_L N_R}$ considering the following squared norm:

$$\|\mathbf{W}_{\text{out}} \mathbf{X} - \mathbf{Y}_{\text{target}}\|_2^2, \tag{3.5}$$

where $\mathbf{Y}_{\text{target}} = [\mathbf{y}(1), ..., \mathbf{y}(N_T)] \in \mathbb{R}^{N_Y \times N_T}$ is the matrix of target values. Thereby, we compute the solution (the values of $\mathbf{W}_{\text{out}}$) that minimizes the norm defined in Equation 3.5 through normal equations approach:

$$\mathbf{W}_{\text{out}} = \mathbf{Y}_{\text{target}} \mathbf{X}^\intercal (\mathbf{X}\mathbf{X}^\intercal + \lambda_r \mathbf{I})^{-1}, \tag{3.6}$$

where $\lambda_r \geq 0 \in \mathbb{R}$ is the Tikhonov regularization term.

Computational Cost of DeepESNs

Since the DeepESN architecture can be seen as a constrained version of the 1-layered ESN (see considerations described in Section 3.2.1), DeepESN presents a considerable lower number of recurrent weights than shallowESN architecture. Considering a total number of recurrent units $N_L N_R$, the cost of computing the state matrix $\mathbf{X}$ in shallowESN architecture is the following:

$$\mathcal{C}_{\text{shallow\_encoding}} = \mathcal{O}(N_L^2 N_R^2 N_T). \tag{3.7}$$

Therefore, the encoding cost of shallowESN increases quadratically with the increasing of the total number of recurrent units. In the case of DeepESN the cost is reduced thanks to the sparsity of the connectivity due to layering constraints:

$$\mathcal{C}_{\text{deep\_encoding}} = \mathcal{O}(N_L N_R^2 N_T). \tag{3.8}$$

Note that, if we consider a constant number of units N_R per reservoir, the encoding cost of DeepESN increases linearly with the increasing of the number of layers N_L.

Asymptotically, the costs of training shallowESN and the DeepESN are equivalent:

$$\begin{aligned} \mathcal{C}_{\text{shallowESN}} &= \mathcal{C}_{\text{shallow\_encoding}} + \mathcal{C}_{\text{readout}} = \mathcal{O}(N_L^2 N_R^2 N_T), \\ \mathcal{C}_{\text{DeepESN}} &= \mathcal{C}_{\text{deep\_encoding}} + \mathcal{C}_{\text{readout}} = \mathcal{O}(N_L^2 N_R^2 N_T). \end{aligned} \tag{3.9}$$

This is due to the dominant term $\mathcal{C}_{\text{readout}}$ in Equations 3.9 (reported in Equation 2.31). However, in practical applications the difference between the encoding costs desctibed in Equations 3.7 and 3.8 can be very relevant, especially if we consider deep recurrent architectures with a low value of N_R. In this thesis, we exploited this efficient approach fixing low values of $N_R \leq 200$ for the considered DeepESN architectures evaluated on real-world applications (see Chapter 5).

Intrinsic Plasticity Pre training in DeepESNs

Here, we introduce a novel pre-training approach for deep RNNs based on the IP algorithm. The idea is to perform progressively the IP pre training (see Equation 2.35) through the layers. The Algorithm 1 show the pre-training procedure to apply the IP learning on deep recurrent architectures.

For each layer l, the Algorithm 1 computes the IP learning as in the following:

- For the first layer, we compute the state starting from the input sequence.

Algorithm 1 IP pre training for DeepESNs (DeepIP)

```
procedure DeepIP(input_sequence)
    for l in 1, ..., N_L do
        x^(l) = computeState(l, input_sequence)        ▷ state on layer l
        g, b = IP(x^(l))                               ▷ IP learning Equation 2.35
        setGainBias(l, g, b)                           ▷ set gain and bias on layer l
        input_sequence = computeState(l, input_sequence)
```

- For the layers with $l > 1$, we compute the state of layer l starting the input sequence computed at pass $l - 1$.
- Given the state $\mathbf{x}^{(l)}$ calculated in one of the two previous points depending on the value l, we compute gain $\mathbf{g}$ and bias $\mathbf{b}$ vectors performing the IP training on $\mathbf{x}^{(l)}$. Then, we set $\mathbf{g}$ and $\mathbf{b}$ vectors on the activation function of layer l (see Equation 2.34). Finally we re-compute the state of layer l exploiting the new bias and gain values. The new state computed is used as input sequence for the next layer.

In the following, we apply always the Algorithm 1 when we perform IP training for DeepESNs.

3.2.1 Architectural Baselines

In this Chapter, the importance of layering in deep RNNs with respect to the construction of a progressively more abstract encoding of the input history, and the relevance of layering per se, are studied through a comparison between different recurrent architectures. Thereby, we introduce two baseline models for the comparison with DeepESN, namely *DeepESN Input to All* (DeepESN-IA) and *GroupedESN*. DeepESN-IA is a DeepESN in which the external input is provided to every layer, while, GroupedESN is characterized by sub-reservoirs fed only by the external input and without a stack organization. Figure 3.2 illustrates the topologies of such architectures.

We define the general state transition function for randomized deep RNNs as following:

$$\mathbf{x}^{(l)}(t) = (1 - a^{(l)})\mathbf{x}^{(l)}(t-1) + a^{(l)} \tanh(\mathbf{W}^{(l)}\mathbf{i}^{(l)}(t) + \hat{\mathbf{W}}^{(l)}\mathbf{x}^{(l)}(t-1)), \tag{3.10}$$

where $\mathbf{W}^{(1)} = \mathbf{W}_{\text{in}}$ and $\mathbf{i}^{(l)}(t)$ is the input of layer l for $l = 1, ..., N_L$. Thereby, the state transition function of DeepESN introduced in Equations 3.1 and 3.2 can be

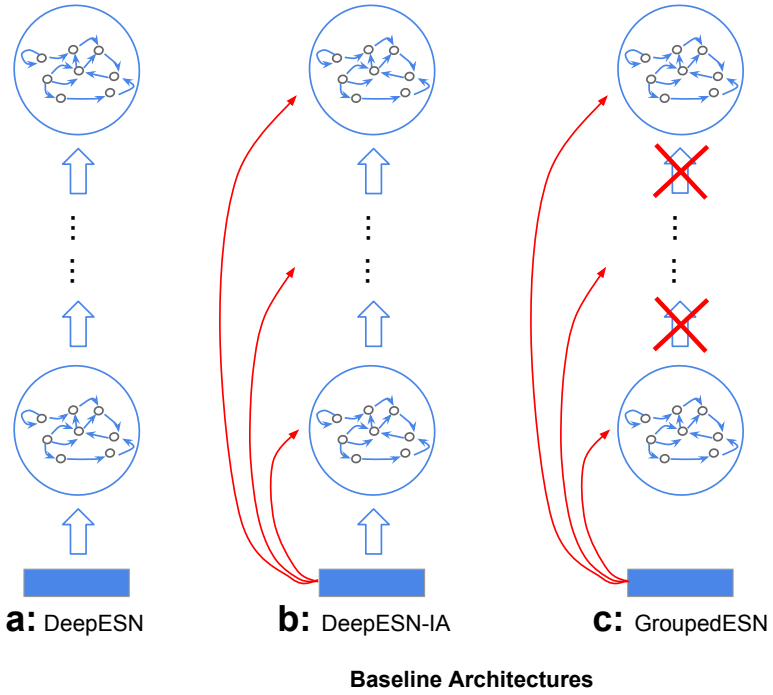


Figure 3.2: Deep RC architectures: (a) DeepESN, (b) DeepESN-IA, (c) GroupedESN. Aspects related to input bias and mathematical notation are not reported here for the ease of graphical representation (see text for details).

expressed by Equation 3.10 in which

$$\mathbf{i}^{(l)}(t) = \begin{cases} \mathbf{u}(t) & if \;\; l = 1 \\ \\ \mathbf{x}^{(l-1)}(t) & if \;\; l > 1. \end{cases} \tag{3.11}$$

Similarly to the case of DeepESN, the state transition function of DeepESN-IA can be expressed by Equation 3.10 in which the input for each layer l $>$ 1 at step t is the concatenation of the external input and the state of the previous layer in the stack:

$$\mathbf{i}^{(l)}(t) = \begin{cases} \mathbf{u}(t) & if \;\; l = 1 \\ \\ \left[\mathbf{u}(t)\ \mathbf{x}^{(l-1)}(t)\right]^T & if \;\; l > 1. \end{cases} \tag{3.12}$$

Accordingly, in a DeepESN-IA for $l > 1$ we have that $\mathbf{W}^{(l)} \in \mathbb{R}^{N_R \times (N_R + N_U)}$. Note that while higher layers in a DeepESN are at increasing distances from the (external) input, in a DeepESN-IA the distance from the input is the same for every layer.

The relevance of layering in DeepESN, with respect to the interplay among the reservoir dynamics at the different levels in the hierarchy, is investigated by considering GroupedESN in which the sub-reservoir are not organized in a

stack. In this case, denoting by l the l-th sub-reservoir, the state transition function of a GroupedESN is defined by setting $\mathbf{i}^{(l)}(t) = \mathbf{u}(t)$ in Equation 3.10 where $\mathbf{W}_{\text{in}}^{(l)} \in \mathbb{R}^{N_R \times N_U}$ for every l, and it can be noticed that the dynamics of sub-reservoirs evolve independently of each other.

As a further architectural baseline, in the following we also take into consideration the case of a standard (shallow) fully connected ESN (whose dynamics are described by Equation 2.21), with the same number of reservoir units as in the whole architecture of a deep RC counterpart.

Note that, critically, a deep layered recurrent network adds architectural constraints to the recurrent connections of a fully connected RNN. A layered RNN can be re-interpreted as a fully connected RNN (with the same number of units) where some connections between groups of neurons are removed. In particular, a layered RNN architecture (see Figure 3.3):

- does not present connections from higher layers to lower layers;
- does not present connections from the input layer to layers at a level greater than one;
- does not present skip connections between layers (each layer is connected only with the successive layer in the pipeline).

Given such considerations, we can consider a layered RNN architecture a constrained version of a fully connected RNN. However, differently by the shallow RNN, the layered RNN does not present time-delays in the flow of the information between consecutive layers. The study of the effects of the aforementioned constraints in the development of the hierarchical temporal dynamics is one of the main subject of this Chapter. Moreover, again under a critical perspective, it is worth to note that an ESN with a shallow reservoir (i.e. without an explicit ordered layered structure) contains already by construction a rich pool of state dynamics (due to the random weight initialization). Hence, the same subject is studied in the following also with respect to the parameters that rule the ESN behavior, in order to investigate the possible enhancement due to a layered structure on their effect and on the state dynamics and temporal representation variety.

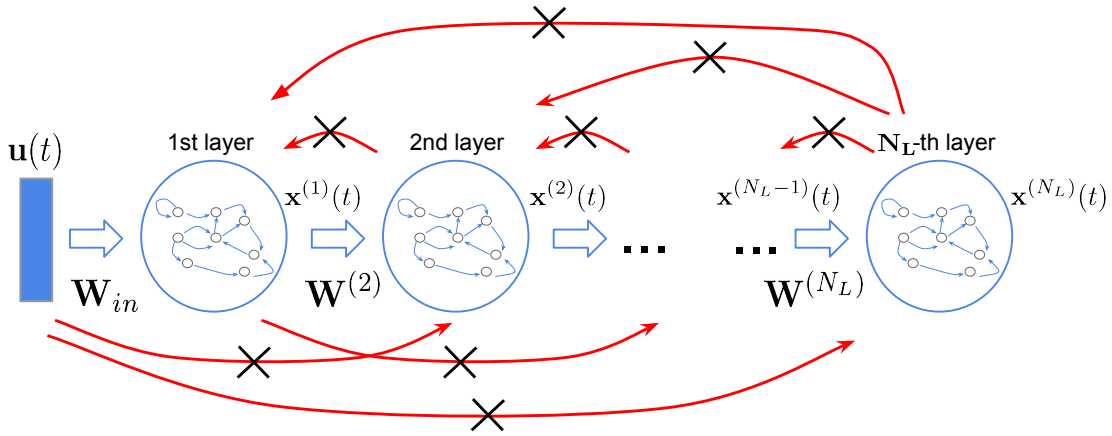


Figure 3.3: The architectural perspective of a deep RNN architecture seen as a constrained RNN.

3.3 Intrinsic Temporal Characterizations of Deep RNNs

In the following we investigate possible strategies aimed at driving the emergence of different time-scales dynamics through the different layers of a deep recurrent architecture.

Our first proposal consists in imposing by design a state dynamics differentiation among the layers, by setting different values of leaky parameter $a^{(l)}$ and spectral radius $\rho^{(l)}$ at different layer l (or sub-reservoirs in GroupedESNs). Using different values of $a^{(l)}$ implies a differentiation among the speed of state dynamics for the different layers of the deep architecture. Indeed, the use of leaky units results in the application of a running average on the state values [72], with the value of the leaky parameter at each layer determining the extent of the persistence of past information in the state dynamics at that layer (in our case, values of $a^{(l)}$ closer to 1 imply that past information is more quickly discarded). In this regard, it is worth to observe that the advantage of having RNN units with different leaky parameters in order to achieve multiple time-scales dynamics has already been discussed in pioneering works already in the 1980s [180, 2]. Varying the values of $\rho^{(l)}$ implies a variation of contractivity [52, 192] and memory length among the state dynamics of different layers. Moreover, as highlighted by Equation 2.25, we rescale the effective spectral radius of the dynamical system in order to control the degree of stability of the state computation [100]. Therefore, the variation of both $\rho^{(l)}$ and $a^{(l)}$ values involves the spectral radius of $\hat{\mathbf{W}}$. Our second proposal consists in using an efficient

unsupervised layer-wise adaptation of reservoir units by means of IP training.

In our experiments, the setting used for the IP training is inline with the literature assessments [195, 178, 171]. In particular, the IP parameters μ, σ and η (see Equation 2.35), are chosen to guarantee the fast convergence, and accordingly, a good stability of the IP algorithm. Moreover, for all the reservoir units we used the same values of the IP parameters and we assessed the influence of layering also in terms of IP effect enhancement.

In this section we present and discuss the results obtained by experimental analysis, on the intrinsic temporal characterization of deep RNNs, conducted by means of numerical simulations on deep RC networks, assessing the effectiveness of the methodologies introduced in Section 3.2.

Specifically, in Section 3.3.1 we investigate the effect of architectural factors on temporal features. In particular, in Section 3.3.1.1 we define qualitative and quantitative measures used for experimental evaluations. In Section 3.3.1.2 we analyze the intrinsic effect of temporal differentiation among layers with the same RC hyper parameters. In Sections 3.3.1.3 we study the differentiation of time scales among the layers caused by the variation of RC hyper parameters. In Sections 3.3.1.4 evaluate the impact of IP on the differentiation of time scales among the dynamics of different layers. In Section 3.3.2, we analyse the distribution of the state developed by the IP algorithm. Moreover, in Section 3.3.3 we further inquire into the impact of IP on the richness of reservoir dynamics in deep RC architectures, and in Section 3.3.4 we evaluate the efficacy of the proposed approaches on the short-term memory capacity of the resulting models.

3.3.1 Multiple Time-scales Differentiation

In order to assess the extent of the time-scales differentiation among the layers in the considered recurrent architectures, similarly to [82] we took into consideration an experimental setting comprising two input sequences, S_1 and S_2, both of length 5000 and identical to each other except for a typo (a perturbation) that is inserted in S_2 at time-step 100. We ran the same RC network on both the unperturbed and the perturbed input sequences and collected the correspondingly obtained reservoir states, evaluating how long the effect of the input perturbation affects the dynamics of each layer by computing the distance between the states corresponding to S_1 and S_2 as a function of time. Specifically, a *qualitative* analysis is provided by plotting the Euclidean distance between the states of corresponding layers in

the unperturbed and perturbed cases. In this concern, note that as S_1 and S_2 are identical until the typo at step $t = 100$, for all the layers the distances among the states are always zero *by construction* for all the time-steps $t < 100$, and are therefore left out from the plots. Moreover, to complete the qualitative results provided by the plots described above, we also adopt *quantitative* measures of time-scales diversification. This is done by assessing the quality of the ordering among the time scales in the different layers by resorting to known distances between rankings [111], i.e. the Kendall's tau (KT) and the Spearman's footrule (SF), and by introducing an index of time-scales separation (IS). Smaller values of KT and SF indicate a better ordering of time scales across the layers, while higher values of IS denote a greater spacing among the duration of the perturbation effect across the layers. Details on these qualitative and quantitative means of investigation (including definitions of KT, SF and IS) are reported in Section 3.3.1.1.

We instantiated this experimental approach by considering two datasets. The first dataset is an *Artificial* time-scales dataset, designed to avoid biases towards specific applications, in which each input element is drawn from a uniform distribution from an alphabet of 10 elements. The second dataset comes from an excerpt of the *Wikipedia* text corpus [184], used in [82] and adopted here to evaluate our results also for the case of a realistic task. Elements in the Wikipedia dataset represent characters, where in our setup we considered an alphabet comprising the 95 most common ones (the printable ASCII characters) and an unknown character (used to represent all the others), as in [82], for a total number of 96 characters. For both the datasets, we represented the input elements by using a one-hot encoding approach, thereby resulting in a one-of-10 encoding for the Artificial dataset and in a one-of-96 encoding for the Wikipedia dataset.

In our experiments, for the only scope of analysis and for the sake of its uniformity and simplicity, we considered DeepESN (and DeepESN-IA) stacked architectures with 10 layers of 10 fully connected units each with input scaling $scale_{\text{in}} = 1$. Analogously, for GroupedESN, we used networks with 10 sub-reservoirs of 10 units. Moreover, for baseline comparison with the standard RC case, we also considered ESNs with 100 fully connected units, i.e. the total number of reservoir units used for the deep RC setup. The models implement 1010 and 9696 free parameters for Artificial and Wikipedia datasets, respectively. For what regards the number of non-trainable weights used in the Artificial dataset, DeepESN and GroupedESN have 2100 parameters, DeepESN-IA has 3090 parameters, while, shallowESN implements 11100 parameters. Concerning the non-trainable weights used in the Wikipedia

dataset, DeepESN has 3046 parameters, DeepESN-IA has 12550 parameters and GroupedESN has 11560 parameters, while, shallowESN implement 19700 parameters. As expected DeepESN, DeepESN-IA and GroupedESN have a fewer number of non-trainable weights than shallowESN model since they are a constrained version of the fully-connected recurrent architecture. This confirms the sparser characterization of the DeepESN architecture w.r.t. shallow counterpart as described in Section 3.2.1. We independently generated 10 guesses for each network hyper-parametrization, and averaged the results over such guesses.

In the following, we present the results of qualitative and quantitative analysis on the Artificial dataset. Moreover, we reported also the quantitative results, i.e. values of KT, SF on the Wikipedia dataset. In the plots presented in the following, the curves for each layer (or sub-reservoir) are averaged over the 10 network guesses considered (in order to avoid influences of single instances on our analysis). Analogously, the values of KT, SF and IS were evaluated on the 10 guesses, reporting min-max ranges for KT and SF, and mean values for IS.

3.3.1.1 Qualitative and Quantitative Measures

Here, we provide details on the measures used in the following to evaluate the goodness of time-scales differentiation among the layers of a stacked RC architecture. Taking into consideration a DeepESN with N_L layers and 2 sequences, the unperturbed one S_1 and perturbed one S_2 (in which a typo is inserted with respect to S_1 at step $t = 100$), here we denote by $\mathbf{x}_u^{(l)}(t)$ and $\mathbf{x}_p^{(l)}(t)$ the state of layer l at step t for the unperturbed and the perturbed sequence, respectively. For each layer l, we evaluated the Euclidean distance between corresponding states $\mathbf{x}_u^{(l)}(t)$ and $\mathbf{x}_p^{(l)}(t)$ as a function of time, i.e. $D^{(l)}(t) = \|\mathbf{x}_u^{(l)}(t) - \mathbf{x}_p^{(l)}(t)\|_2$. Then we plotted the distances $D^{(l)}(t)$ for $t \geq 100$ and for all the layers, in order to graphically investigate how long the effect of the input perturbation at step 100 affects the state dynamics of each layer, providing a *qualitative* analysis of the time-scales differentiation emerging in the architecture. Analogous plots can of course be obtained also for the cases of deepESN-IA and GroupedESN.

The qualitative investigation described above is completed by adopting *quantitative* measures of time-scales diversification. To this aim, the maximum duration of the perturbation effect on layer l can be expressed as $P^{(l)} = \max_t(D^{(l)}(t) > 0)$. By ordering the set of values $\{P^{(l)}\}_{l=1}^{N_L}$, we can define a ranking on the layers based on the duration of the perturbation effect, denoted by $\{O^{(l)}\}_{l=1}^{N_L}$, and which represents a permutation of $\{1, 2, \ldots, N_L\}$. Specifically, if $O^{(l)} = n$ it means that the layer l is

the n-th one in terms of duration of the input perturbation effect. In this sense, the ideal case is represented by the identity permutation ranking $1, 2, ..., N_L$, i.e. $O^l = l$ for every $l = 1, \ldots, N_L$, corresponding to an increasing duration of the perturbation effect for higher layers. Based on these definitions, we can quantify the quality of the ordering among the time scales in the network's layers by measuring the distance between the ranking $\{O^{(l)}\}_{l=1}^{N_L}$ and the identity permutation ranking. To do so, we adopt two known distances between rankings [111], i.e. the Kendall's tau and the Spearman's footrule distances, respectively denoted by KT and SF, and computed according to:

$$\begin{aligned} KT &= |\{(l_1, l_2) : (1 \leq l_1 < l_2 \leq N_L) \wedge (O^{(l_1)} > O^{(l_2)})\}|, \\ SF &= \textstyle\sum_{l=1}^{N_L} |l - O^{(l)}|, \end{aligned} \tag{3.13}$$

where, with respect to the identity permutation, KT sums the number of required pairwise swaps, while SF sums the total amount of displacement of the elements in the ranking. Accordingly, smaller values of KT and SF denote better orderings of the times-scales.

Moreover, we can quantify the extent of time-scales separation by measuring the distances between the duration of the perturbation in consecutive layers, introducing an index of separation, denoted by IS, and computed as:

$$IS = \sum_{l=2}^{N_L} P^{(l)} - P^{(l-1)}, \tag{3.14}$$

where higher values of IS correspond to a greater spacing among the duration of the perturbation effect in the different layers.

3.3.1.2 Intrinsic Architectural Differentiation

Our experimental analysis on the multiple time-scales differentiation is conducted by firstly considering fixed values of the leaky parameter $a = 0.55$ and of the spectral radius $\rho = 0.9$ in which $a = a^{(l)}$ and $\rho = \rho^{(l)}$ for each layer (or sub-reservoirs) l. Note that ρ and a values are intentionally not optimized, as the purpose is not to achieve the best results, but to show the differences occurring among the different architectures under the same conditions. Figure 3.4 graphically shows the results achieved on the Artificial dataset with DeepESN, DeepESN-IA and GroupedESN. Continuous blue lines refer to the different layers of the deep architecture (different

sub-reservoirs for GroupedESN), with darker colors corresponding to higher layers. For the sake of comparison, the red dashed line refers to the shallowESN baseline with $a = 0.55$ and $\rho = 0.9$, as in every layer of the deep networks. Table 3.1 reports the values of KT, SF and IS achieved by DeepESN, DeepESN-IA and GroupedESN on both the Artificial and the Wikipedia datasets.

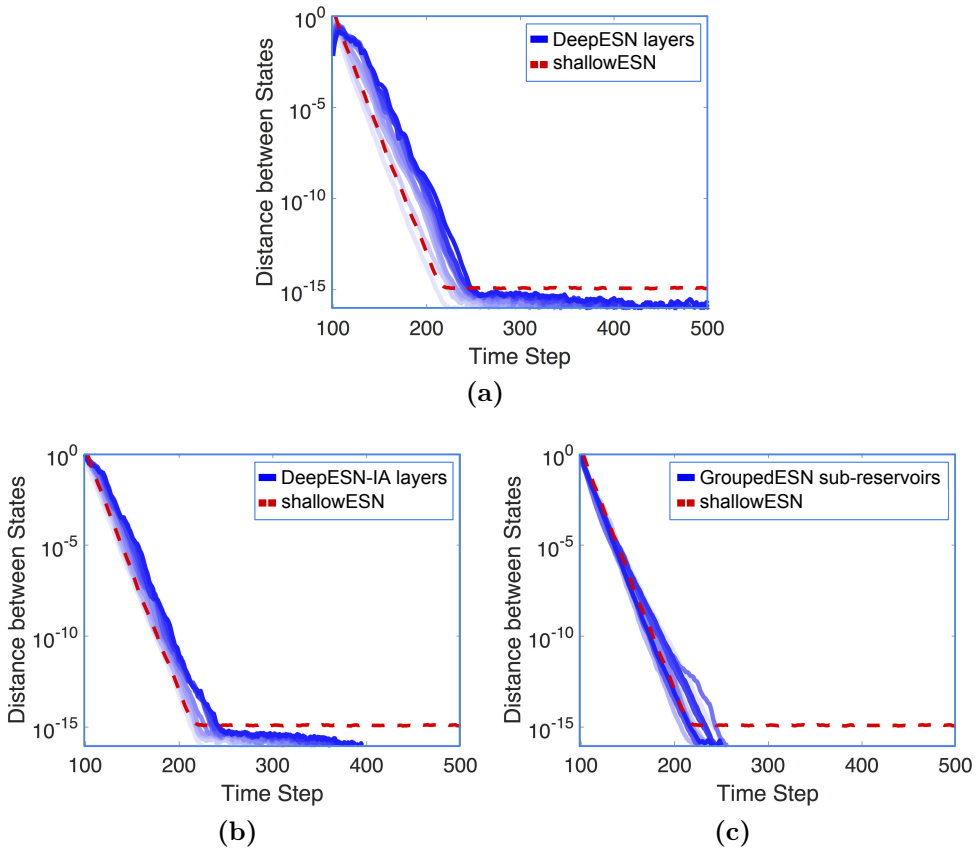


Figure 3.4: Distance between perturbed and unperturbed states on the Artificial dataset for the considered RC architectures with $a = 0.55$ and $\rho = 0.9$ for every layer (or sub-reservoir). Continuous blue lines correspond to layers in deep RC networks (sub-reservoirs in GroupedESN), dashed red lines correspond to the shallowESN with the same number of total reservoir units and hyper-parametrization. **a**: DeepESN, **b**: DeepESN-IA, **c**: GroupedESN.

The intrinsic differentiation among the time-scales dynamics at the different layers of a DeepESN is qualitatively analyzed through the plot shown in Figure 3.4a, from which it is possible to observe that the effects of the input perturbation last longer for higher layers in the stack. Such differentiation is indeed related to the layered deep architecture, as it is strongly attenuated when the external input is provided to each layer, as in the case of DeepESN-IA (see Figure 3.4b) or when layering is removed from the architectural design, as in the case of GroupedESN (see Figure 3.4c). These insights are quantitatively confirmed by the results in Table 3.1,

showing that DeepESN provides a preferable differentiation of time scales at the different layers, in this regard generally presenting a more ordered organization of time scales, i.e. smaller values of KT and SF, and better separability, i.e. larger values of IS.

In particular, a comparison between the behaviors of DeepESN and DeepESN-IA points out the relevance of having higher layers at increasing distances from the external input as a key architectural factor for time-scales separation. Note that DeepESN and DeepESN-IA show a similar hierarchical organization of time scales (similar values of KT and SF) and in both cases the higher layers of the architecture present longer time scales than the corresponding standard ESN (as can be seen in the plots of Figures 3.4a and 3.4b). However DeepESN-IA shows a reduced separation of time scales with respect to DeepESN, as can be seen graphically through a comparison of Figure 3.4a and Figure 3.4b, and also numerically in Table 3.1, with DeepESN-IA leading to smaller IS values than DeepESN. From a theoretical point of view, the time-scales ordering with progressively slower dynamics highlighted in Figure 3.4a is due to the progressive decreasing of the contractivity in higher layers dynamics [55].

The case of GroupedESN, illustrated in Figure 3.4c, shows the intrinsic variability that can be already present in (sub-)reservoirs with the same hyper-parametrization when they are not organized in a stack. As can be seen, in this case the dynamics of all the sub-reservoirs do not present a particular ordering and have a similar behavior to the one of the shallowESN with corresponding total number of units and values of the hyper parameters. Quantitatively, a comparison between the results of DeepESN and GroupedESN in Table 3.1, shows the inherent impact in terms of ordering and separability among the time-scales dynamics (smaller values for KT and SF, larger values of IS) that are due to the hierarchical organization of the reservoir layers in DeepESN. Notice that in this setting, in which there are no hierarchies among the sub-reservoirs of a GroupedESN (they all have the same hyper-parametrization), the use of different grades of colors in Figure 3.4c and the values of KT, SF and IS reported in Table 3.1 for GroupedESN assume a different meaning than in the case of layered architectures. The results of GroupedESN in this case is indeed representative of a completely un-ordered sub-reservoir organization and are therefore reported for the sake of completeness and scale comparison.

Model	KT (min-max)	SF (min-max)	IS (mean)
Artificial			
DeepESN $a = 0.55$	0 - 2	0 - 2	203.90 ($\pm$83.39)
DeepESN-IA $a = 0.55$	0 - 2	0 - 2	134.10 ($\pm$37.68)
GroupedESN $a = 0.55$	8 - 10	26 - 42	18.70 ($\pm$56.56)
Wikipedia			
DeepESN $a = 0.55$	0 - 2	0 - 2	175.70 ($\pm$98.48)
DeepESN-IA $a = 0.55$	0 - 4	0 - 4	123.90 ($\pm$22.48)
GroupedESN $a = 0.55$	7 - 10	22 - 44	-22.40 ($\pm$58.16)

Table 3.1: Values of Kendall's tau (KT), Spearman's footrule (SF) and index of separation (IS) achieved on the Artificial dataset and on the Wikipedia dataset by DeepESN, DeepESN-IA and GroupedESN with with $a = 0.55$ and $\rho = 0.9$ for every layer (or sub-reservoir). For KT and SF smaller values are better, for IS higher values are better.

3.3.1.3 Differentiation by Variation of RC Hyper parameters

In light of the results shown in Section 3.3.1.2, we can observe that the inherent diversification among the layers dynamics in a DeepESN is quite narrow (Figure 3.4a), with the range of emerging time scales presenting a limited extent. Such differentiation can be emphasized within the efficient RC approach by resorting to the strategies described at the beginning of Section 3.3.

Variation of leaky parameter

We first take into consideration the effect due to a diversification of the value of the leaky parameter among the layers. Figure 3.5 shows the results achieved by DeepESN, DeepESN-IA and GroupedESN using a fixed value of $\rho = 0.9$ and decreasing values of the leaky parameter a for increasing layer depth, from 1 to 0.1, thus imposing a progressively slower speed of reservoir dynamics at higher layers in the architecture. Table 3.2 reports the KT, SF and IS values obtained by DeepESN, DeepESN-IA and GroupedESN in the same conditions.

For the sake of comparison, in each plot of Figure 3.5 it is reported also the result obtained by standard shallowESN with values of $\rho = 0.9$, as in every layer of the deep RC networks, and $a = 0.55$, i.e. the average value among the layers of the deep architectures. Such result is reported here (and also in the following analysis) as a summary for the values and comparisons already discussed with regard to Figure 3.4, as indeed the aim is to assess the extent of the differentiation among the behaviors shown by the different layers, also in comparison to the average case.

As can be seen in Figure 3.5a, the variability of the leaky parameter has a great

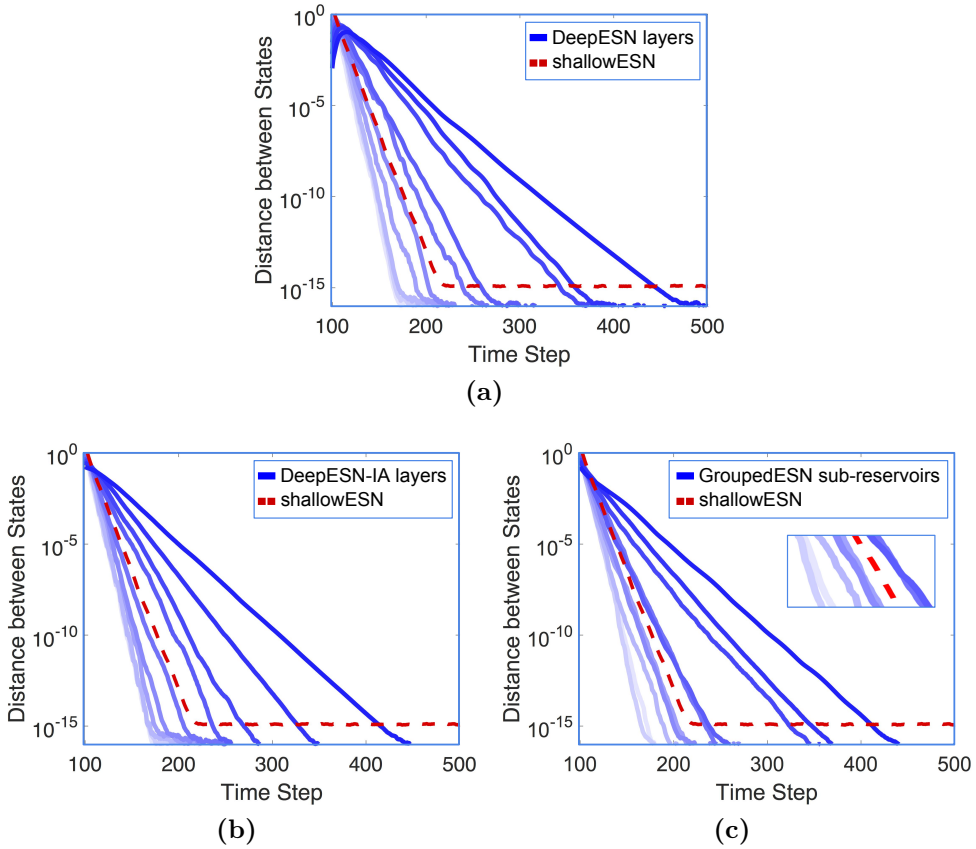


Figure 3.5: Distance between perturbed and unperturbed states on the Artificial dataset for the considered RC architectures with $\rho = 0.9$ for every layer (or sub-reservoir) and a varying from 1 to 0.1 among the layers (or sub-reservoirs). Continuous blue lines correspond to layers in deep RC networks (sub-reservoirs in GroupedESN), dashed red lines correspond to the shallowESN (for graphical reference with respect to Figure 3.4, see text). **a**: DeepESN, **b**: DeepESN-IA, **c**: GroupedESN.

impact on the differentiation among the emerging time-scales dynamics, showing a much wider extent of the ordered diversification than DeepESN with fixed values of a (Figure 3.4a). As can be seen by comparing Tables 3.2 and 3.1, varying the value of the leaky parameter across the layers of a DeepESN results in generally lower values of KT and SF and higher values of IS.

This characterization is a result of the interplay between layering and leaky integration variability, and also in this case it is strongly reduced when all the layers are at the same distance from the input, i.e. for DeepESN-IA, or when non-stacked architectures are considered, i.e. for GroupedESN. Specifically, also in this case, DeepESN-IA leads to a reduced separation of time scales across the layers, while GroupedESN in addition to the reduced separation also results in a worse ordering with respect to the duration of the perturbation effect, which is graphically pointed

out by the overlapping among the curves in Figure 3.5c (highlighted in the zoom), and by the results in Table 3.2.

Model	KT (min-max)	SF (min-max)	IS (mean)
Artificial			
DeepESN	0 - 0	0 - 0	367.80 (±76.25)
DeepESN-IA	0 - 2	0 - 2	294.30 (±44.51)
GroupedESN	2 - 9	4 - 18	285.00 (±50.07)
Wikipedia			
DeepESN	0 - 2	0 - 2	335.50 (±92.69)
DeepESN-IA	0 - 2	0 - 2	295.00 (±42.54)
GroupedESN	4 - 9	4 - 18	298.10 (±48.86)

Table 3.2: Values of Kendall's tau (KT), Spearman's footrule (SF) and index of separation (IS) achieved on the Artificial dataset and on the Wikipedia dataset by DeepESN, DeepESN-IA and GroupedESN with with $\rho = 0.9$ for every layer (or sub-reservoir) and a varying from 1 to 0.1 among the layers (or sub-reservoirs). For KT and SF smaller values are better, for IS higher values are better.

Variation of spectral radius

Here we provide the results on time-scales differentiation due to the variability of the spectral radius ρ among the layers of a stacked RC network. Figure 3.6 shows the results achieved by DeepESN, DeepESN-IA and GroupedESN using a fixed value of $a = 0.55$ and increasing values of the spectral radius ρ for increasing layer depth, from 0.1 to 0.9[1], resulting in increasing memory length for higher layers. In Figure 3.6 we also show the result of standard shallowESN with a and ρ equal to the corresponding averages among the layers of the deep architectures. Table 3.3 reports the KT, SF and IS values obtained by DeepESN, DeepESN-IA and GroupedESN in the same conditions. For the sake of reference comparison, Figure 3.6 and Table 3.3 also report the results obtained by DeepESN with constant value of $\rho = 0.5$, i.e. the average among the ρ values in the considered range of variability.

The effect of the spectral radius variation can be appreciated by comparing the results obtained for the cases of DeepESN with constant ρ for every layer (Figure 3.6a) and of DeepESN with ρ varying among the layers (Figure 3.6b). As can be seen, varying the value of ρ leads to an improvement of the hierarchical time-scales differentiation, as also reflected by the values in Table 3.3, with DeepESN

[1]The ρ values relative to the layers are 10 evenly spaced points between 0.1 and 0.9.

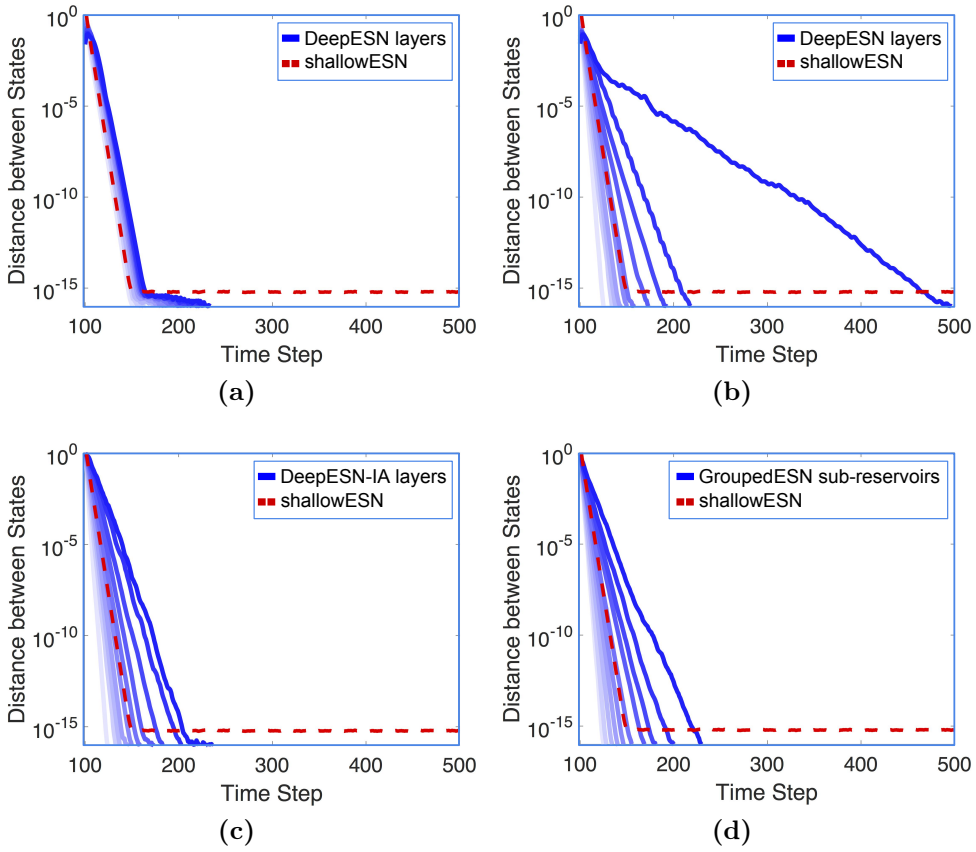


Figure 3.6: Distance between perturbed and unperturbed states on the Artificial dataset for the considered RC architectures with $a = 0.55$ for every layer (or sub-reservoir) and ρ varying from 0.1 to 0.9 among the layers (or sub-reservoirs). Continuous blue lines correspond to layers in deep RC networks (sub-reservoirs in GroupedESN), dashed red lines correspond to the shallowESN (for graphical reference with respect to Figure 3.4, see text). For the sake of reference the results corresponding to DeepESN with constant $\rho = 0.5$ for all the layers is reported as well. **a**: DeepESN with constant ρ, **b**: DeepESN, **c**: DeepESN-IA, **d**: GroupedESN.

using different values of ρ achieving better results than DeepESN with constant ρ in terms of KT, SF and IS values.

Comparing the results showed in Tables 3.3 and 3.2, we can note that in the case of variable ρ the effect of time-scales differentiation is less significant for all architectures. In particular, the DeepESN with variable ρ obtained a high standard deviation of IS value on the Artificial dataset. This can compromise the improving of the IS value in this particular case. Anyway, DeepESN obtained good (low) KT and SF values on the Artificial dataset (see Table 3.3). This means that the quality of the time-scales ordering among the layers is empirically preserved also for eventual non-optimal IS values.

Model	KT (min-max)	SF (min-max)	IS (mean)
Artificial			
DeepESN $\rho = 0.5$	0 - 3	0 - 4	68.20 ($\pm$21.16)
DeepESN var. ρ	0 - 2	0 - 2	161.90 ($\pm$129.19)
DeepESN-IA var. ρ	0 - 4	0 - 4	95.20 ($\pm$24.02)
GroupedESN var. ρ	0 - 6	0 - 6	95.40 ($\pm$22.57)
Wikipedia			
DeepESN $\rho = 0.5$	0 - 6	0 - 6	54.90 ($\pm$17.43)
DeepESN var. ρ	0 - 2	0 - 2	168.00 ($\pm$69.95)
DeepESN-IA var. ρ	0 - 2	0 - 2	92.80 ($\pm$18.75)
GroupedESN var. ρ	0 - 4	0 - 6	100.70 ($\pm$39.86)

Table 3.3: Values of Kendall's tau (KT), Spearman's footrule (SF) and index of separation (IS) achieved on the Artificial and on the Wikipedia datasets by DeepESN, DeepESN-IA and GroupedESN with $a = 0.55$ for every layer (or sub-reservoir) and ρ varying from 0.1 to 0.9 among the layers (or sub-reservoirs). For the sake of reference comparison the results achieved for the case of DeepESN with constant $\rho = 0.5$ for all the layers is reported as well. For KT and SF smaller values are better, for IS higher values are better.

3.3.1.4 Differentiation by IP Training

The impact on the development of multiple time scales due to the unsupervised IP training is shown in Figure 3.7 and Table 3.4, considering the cases of DeepESN, DeepESN-IA and GroupedESN with constant values of $a = 0.55$ and $\rho = 0.9$ for all the layers (or sub-reservoirs), and using IP learning. In our experimental setting, we used values of $\mu = 0$, $\sigma = 0.1$ and $\eta = 0.00001$ for the IP parameters in Equation 2.35. For comparison, in the plots of Figure 3.7 we also show the result obtained by the corresponding standard shallowESN architecture using IP with and the same hyper-parametrization.

The remarkable effect of IP on the time-scales differentiation in a layered architecture is pointed out by a comparison between the results of DeepESN under the same settings of a and ρ, with IP learning (Figure 3.7a) and without IP learning (Figure 3.4a). It can be observed that after IP training the higher layers in the DeepESN architecture tend to forget more slowly the past input history, and the effect of the typo perturbation has a much longer duration. Results in Table 3.4 show that DeepESN with IP achieves better results in terms of time-scales ordering and separation among the layers, outperforming the results of the base DeepESN case with corresponding settings (in Table 3.1).

In addition to the amplifying effect of IP on the time-scales differentiation observed on DeepESN, it is also possible to notice the enhancement effect of layering on the

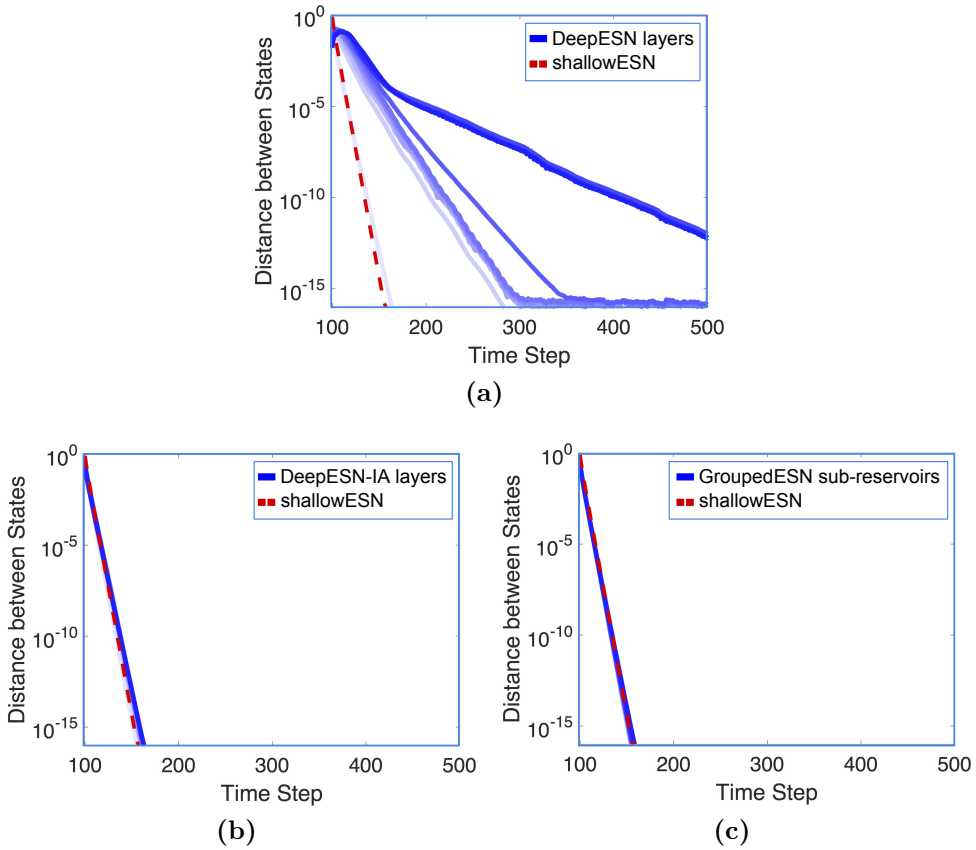


Figure 3.7: Distance between perturbed and unperturbed states on the Artificial dataset for the considered RC architectures, with $a = 0.55$ and $\rho = 0.9$ for every layer (or sub-reservoir) and using IP learning. Continuous blue lines correspond to layers in deep RC networks (sub-reservoirs in GroupedESN), dashed red lines correspond to the shallowESN with the same number of total reservoir units and hyper-parametrization. **a**: DeepESN, **b**: DeepESN-IA, **c**: GroupedESN.

IP efficacy. Indeed the hierarchical organization of reservoir layers in DeepESN, with higher layers at increasing distance from the input, allows to trigger a process of increasing effectiveness of IP among the layers, as can be seen also by the fact that the curves representing the dynamics of the first DeepESN layer and of the shallowESN almost overlap in the plot in Figure 3.7a. On the other hand, when the DeepESN architectural characterizations are lost, layer dynamics are made more uniform by IP learning, as can be seen for DeepESN-IA (Figure 3.7b) and GroupedESN (Figure 3.7b).

The strong effect of IP on the emerging of multiple time-scales differentiation in DeepESN can be explained in terms of a diversification of the memory length in the different layers, similarly to the effect of the variation of the spectral radius. Indeed, by changing the gains of the reservoir units' activation functions, IP potentially act on the real value of the spectral radius at the different layers, as noticed also in

Model	KT (min-max)	SF (min-max)	IS (mean)
Artificial			
DeepESN	0 - 2	0 - 2	785.10 (±248.97)
DeepESN-IA	0 - 7	0 - 14	24.00 (±8.99)
GroupedESN	6 - 10	20 - 44	-0.40 (±3.23)
Wikipedia			
DeepESN	0 - 0	0 - 0	644.40 (±183.61)
DeepESN-IA	2 - 8	2 - 14	20.90 (±4.66)
GroupedESN	6 - 10	10 - 44	-1.80 (±6.84)

Table 3.4: Values of Kendall's tau (KT), Spearman's footrule (SF) and index of separation (IS) achieved on the Artificial dataset and on the Wikipedia dataset by DeepESN, DeepESN-IA and GroupedESN, with with $a = 0.55$ and $\rho = 0.9$ for every layer (or sub-reservoir) and using IP learning. For KT and SF smaller values are better, for IS higher values are better.

[178] for standard RC networks.

Although the effect of IP deserves a separate theoretical analysis, we can provide an insight about theoretical motivations on the effect illustrated in Figure 3.7a). In particular, the IP algorithm modifies the gains of the layers progressively moving the effective spectral radius of the recurrent weights close to the edge of stability [171]. Therefore, the contractivity of the transition function is progressively decreased determining higher layers with higher memory.

3.3.2 Effects of IP in the distribution of DeepESN dynamics

Here, we qualitatively analyze the effects of the IP training on the distribution of the state signals developed by layers in deep recurrent architectures. Similarly to the study of distributions in shallow reservoirs presented in [178], for each layer we plotted the histogram of a neuron signal in order to highlight the distribution through the layers. For these experiments, we considered a DeepESN architecture with the same configurations used in the previous Section 3.3.1.4.

Figure 3.8 shows the distribution developed through the layers states by DeepESN with and without IP. As we expected, the IP algorithm allows the layered architecture to progressively produce state signals with distributions that match the target Gaussian ones all centered on zero as illustrated in Figure 3.8 by red bins. Instead, in the case of state signals developed without IP, each layer has a different distribution (represented by azure bins in Figure 3.8). Moreover, in higher layers (layer 4 and layer 7) the values of the azure distributions are focused far from zero on a few points. This indicates that without IP training the state signal can saturate with

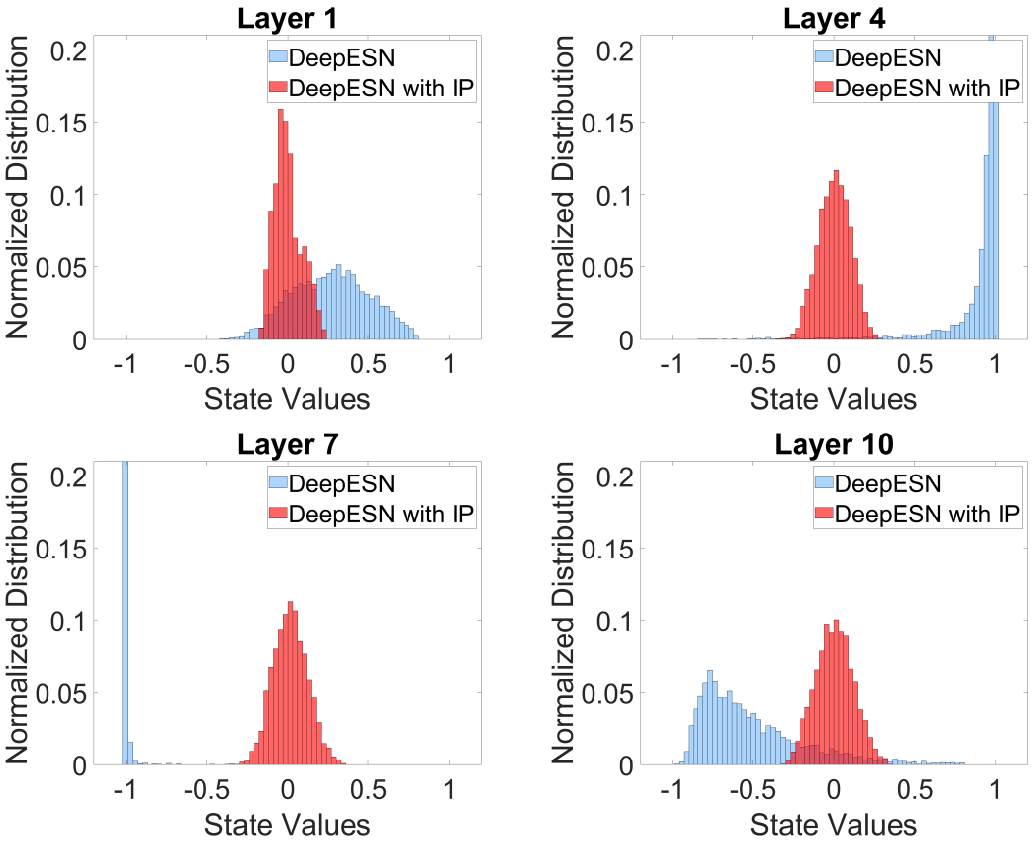


Figure 3.8: Distribution of the state signals developed progressively in higher layers of DeepESN with (red bins) and without (azure bins) IP, Top Left: layer 1, Top Right: layer 4, Bottom Left: layer 7, Bottom Right: layer 10.

the risk of producing poor temporal representations. Overall, our results indicate the practical beneficial effects of IP training, which allows the DeepESN architecture to enrich the temporal representation maximizing the entropy (i.e., having Gaussion distributions), and at the same time avoiding the saturation of temporal features.

In the next section, we measure the richness of the temporal features developed by a DeepESN with IP training in terms of Shannon's differential entropy.

3.3.3 Richness of Reservoir Dynamics: IP Training and Layering

The role of IP learning in relation to layering then deserves to be further investigated in the context in which it has been introduced, i.e. the information maximization of reservoir state dynamics [171, 178, 195]. To this aim, we evaluated the entropy of reservoir units activations over time, as a measure of the richness of state dynamics, assessing the effect of IP in conjunction with layering. We

approximated the entropy of the output distribution of each reservoir unit i, by computing the integral estimate [12] H_i:

$$H_i = -\int \texttt{pdf}_i(x) \log \texttt{pdf}_i(x) dx \tag{3.15}$$

where $\texttt{pdf}_i$ is the estimate of the probability density function of the i-th reservoir unit output distribution over time, computed by means of kernel density estimation, and the integral in Equation 3.15 is computed by numerical integration[2]. Algorithm 2 describe the computation of the state entropy used in the following experiments.

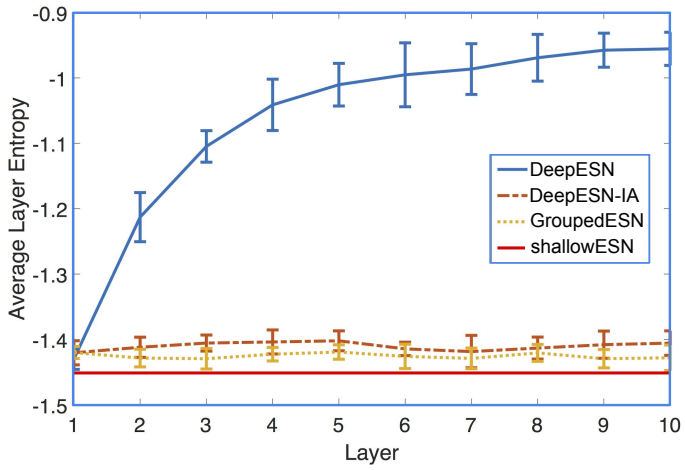
Algorithm 2 Entropy Computation

```
procedure ENTROPY(x^(l), i)
    p = fitdist(x^(l)(i,:), 'kernel')        ▷ pdf_i by MATLAB function fitdist
    f = @(x) p.pdf(x^(l)(i,:)).*log(p.pdf(x^(l)(i,:)))
    H = -integral(f,-10,10)                  ▷ compute Equation 3.15
    return H
```

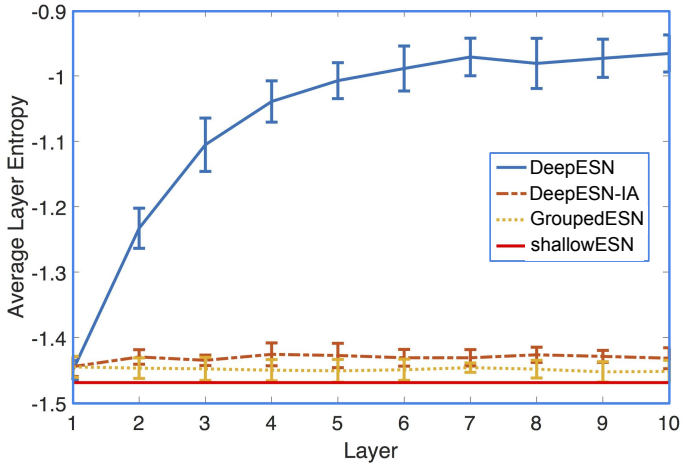
The layer-wise effect of IP on the entropy of reservoir units activations is graphically shown in Figure 3.9 for both the Artificial (Figure 3.9a) and the Wikipedia (Figure 3.9b) datasets. The plots show the values of the entropy averaged on the units of each layer (or sub-reservoir) for DeepESN, DeepESN-IA and GroupedESN, using the same experimental setting considered with regard to Figure 3.7 and Table 3.4 (i.e. $a = 0.55$, $\rho = 0.9$, $\mu = 0$ and $\sigma = 0.1$). For the sake of comparison, Figure 3.9 also shows the entropy achieved by a shallowESN (with the same total number of reservoir units) under the same conditions. It can be seen that for both the datasets, the entropy of DeepESN states clearly increases with increasing layer depth, showing the same incremental IP effect already observed in Figure 3.7a. On the other hand, in the cases of DeepESN-IA and GroupedESN, the entropy remains almost constant among the layers (or sub-reservoirs), and very close to the values corresponding to a standard shallowESN.

The values of the state entropy averaged over all the reservoir units in DeepESN, DeepESN-IA, GroupedESN and ESN in the same experimental settings are reported in Table 3.5. From such results it is possible to appreciate the overall strong impact of IP on the hierarchical architecture of DeepESN, resulting in an average entropy improvement of $\approx 27\%$ with respect to the shallowESN for both the datasets, whereas DeepESN-IA and GroupedESN obtained results very close to those of

[2]Note that the H_i values computed by means of Equation 3.15 result in approximations of the Shannon's differential entropy, which can also assume negative values.



(a)



(b)

Figure 3.9: Layer-wise averaged entropy of reservoir states on the Artificial dataset **(a)** and on the Wikipedia dataset **(b)**, computed for DeepESN, GroupedESN and DeepESN-IA, with $a = 0.55$ and $\rho = 0.9$ for every layer (or sub-reservoir) and IP learning. For GroupedESN the results refer to sub-reservoirs. The average entropy of the shallowESN counterpart is reported as a continuous red line across each plot.

shallowESN. Results in Table 3.5 confirm that in this experimental setting the effectiveness of IP is enhanced only by using a hierarchical reservoir organization with layers at increasing distance from the input. When IP is applied to layers of reservoir units at the same distance from the input, or to non-stacked sub-groups of reservoir units, analogous results to the application of IP to a shallowESN are achieved.

Model	Entropy
Artificial	
DeepESN	-1.066 ± 0.021
DeepESN-IA	-1.410 ± 0.007
GroupedESN	-1.425 ± 0.005
shallowESN	-1.451 ± 0.003
Wikipedia	
DeepESN	-1.071 ± 0.019
DeepESN-IA	-1.431 ± 0.006
GroupedESN	-1.449 ± 0.004
shallowESN	-1.469 ± 0.005

Table 3.5: Average entropy of reservoir states on the Artificial and on the Wikipedia datasets (higher values are better), obtained by DeepESN, GroupedESN and DeepESN-IA,with $a = 0.55$ and $\rho = 0.9$ for every layer (or sub-reservoir) and IP learning. The average Entropy of corresponding shallowESN is reported as well for the sake of reference comparison.

3.3.4 Short-term Memory Capacity

A last set of experiments has been considered to assess the effectiveness of the proposed approaches on the MC task [97]. This task provides a measure of short-term memory capacity of RC networks, by evaluating how well it is possible to recall delayed versions of the input based on reservoir activations. Input consists of a temporal signal whose elements $u(t)$ are drawn from a uniform distribution over $[-0.8, 0.8]$. The task requires to reconstruct the input stream with increasing delays, i.e. for each time-step t we consider target values $\bar{y}_k(t) = u(t-k)$, for $k = 0, ..., \infty$. The overall MC is defined as:

$$MC = \sum_{k=0}^{\infty} r^2(u(t-k), y_k(t)), \tag{3.16}$$

where $r^2(u(t-k), y_k(t))$ is the squared correlation coefficient between the input with delay k and the corresponding re-constructed value $y_k(t)$. The estimation of the MC value in Equation 3.16 can be numerically unstable. In the case of linear architectures, it is possible to theoretically estimate the MC value with a high degree of precision [154], while, the estimation is more difficult in the general case with non-linear architectures. However in the practice, due to theoretical results on RC networks [97], the MC can be computed by considering only a finite number of delayed signals. Here, we set up an MC task similarly to [171], by considering a number of delays equal to 200 (i.e. twice the number of total reservoir

units considered). The input signal contained 6000 steps, 5000 of which used for training and the remaining 1000 for test. For this task we adopted similar settings to those already used for previous experiments. In particular, we considered DeepESN architectures with 10 reservoir layers of 10 fully connected units and input scaling $scale_{\text{in}} = 0.1$[3], instantiating the networks with constant values of the leaky parameter $a \in \{0.1, 0.55, 1\}$ and of the spectral radius $\rho \in \{0.1, 0.5, 0.9\}$ among the layers. Moreover we considered DeepESN settings in which the value of a varies from 1 to 0.1 among the layers, with constant values for $\rho \in \{0.1, 0.5, 0.9\}$, and in which the value of ρ varies among the layers from 0.1 to 0.9, with constant values of $a \in \{0.1, 0.55, 1\}$. We also ran experiments using IP learning, which has a known improvement effect on the MC task [171], using values of $\eta = 0.00001$, $\mu = 0$ and $\sigma \in \{0.1.0.01\}$ for all the RC settings mentioned above. Analogous experiments were conducted for DeepESN-IA and GroupedESN, as well as for standard ESN (for the sole scope of baseline reference and assessment). For each network hyper-parametrization, we independently generated 10 guesses, averaging the results over such guesses. In all the considered cases, the values of a, ρ and σ were chosen by model selection on a validation set (comprising 20% of the data in the training set).

The MC values on the test set achieved by DeepESN, DeepESN-IA, GroupedESN and shallowESN are reported in Table 3.6. Results show that DeepESN obtained the best MC both without IP and with IP, improving the results obtained by shallowESNs (which are in line with literature results [171]). In particular, without IP, DeepESN obtained an MC value of 42.45 that represents an improvement of $\approx 54\%$ with respect to the value achieved by shallowESN. The hierarchical organization of DeepESN architecture also allowed to exalt the known effect of IP on the MC, leading to a value of 54.49, which improves the result achieved by shallowESN with IP by $\approx 47\%$. In both the cases, without and with IP, the selected values of spectral radius and leaky parameter for DeepESN were $a = 1$ and $\rho = 0.9$, while varying the values of these two parameters among the layers led to slightly lower MC results. Moreover, notice that DeepESN-IA and GroupedESN achieved MC values very close to the one of shallowESN, both with and without IP.

A further comparison between the MC of DeepESN and shallowESN is presented in Figure 3.10, which shows the results achieved in correspondence of different values of the leaky parameter a and of the spectral radius ρ (constant for all the

[3]Some works show that lower values of $scale_{\text{in}}$ can obtain better MC results [44, 45], however, since we are interested to the study of the effect of layering on MC, we focus on a single $scale_{\text{in}}$ value.

Model	Memory Capacity
DeepESN	42.45 ± 3.11
DeepESN + var. a	37.15 ± 2.48
DeepESN + var. ρ	30.79 ± 1.15
DeepESN-IA	28.05 ± 1.87
GroupedESN	28.02 ± 1.77
shallowESN	27.50 ± 1.34
DeepESN + IP	54.49 ± 3.82
DeepESN + var. a + IP	52.03 ± 5.43
DeepESN + var. ρ + IP	48.01 ± 3.36
DeepESN-IA + IP	36.78 ± 2.69
GroupedESN + IP	39.02 ± 2.25
shallowESN + IP	37.06 ± 1.48

Table 3.6: Memory Capacity results (higher is better) achieved by DeepESN, DeepESN-IA and GroupedESN and shallowESN. Results for DeepESN are reported also for the cases of layers with decreasing values of a (var. a) and increasing values of ρ (var. ρ) among the layers. The first group of results refers to RC models without the use of IP, the second group refers to the corresponding models with IP (denoted by +IP).

layers) while the values of the other parameters were selected on the validation set, without using IP (Figures 3.10a and 3.10b) and using IP (Figures 3.10c and 3.10d). Results clearly show that DeepESN improves the short-term MC of shallowESN in all the cases.

Overall, as highlighted by the experiments reported in Section 3.3.1.2, the DeepESN is able to develop a hierarchical multiple time-scales representation of the driving input ordered among the layers. This leads to increasing the memory capacity of the higher layers. Therefore, higher architectures can globally exploit a better memory capacity regardless of the temporal structure of the input sequence.

3.4 Hierarchical and Distributed Temporal Representation in DeepRNNs

In the previous section 3.3 we highlighted the role of layering in deep RNNs in developing progressively different temporal dynamics among the layers. In this section, we take a step forward in the study of the structure of the temporal features naturally emerging in layered RNNs. To this aim, we resort to classical tools in the area of signal processing to analyze the differentiation among the state representations developed by the different levels of a DeepESN on signals

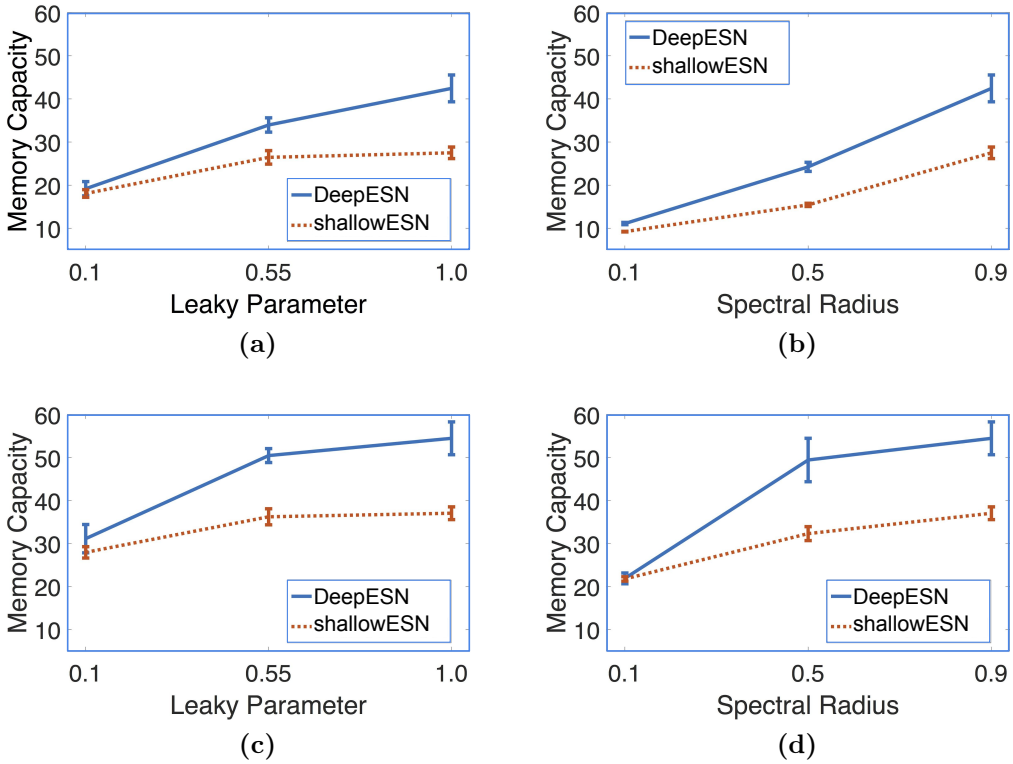


Figure 3.10: MC results of DeepESN and shallowESN for different values of the spectral radius ρ and of the leaky parameter a. **(a)**: different values of a, without IP, **(b)**: different values of ρ, without IP, **(c)**: different values of a, with IP, **(d)**: different values of ρ, with IP. For DeepESN each reservoir layer has the same hyper-parametrization.

characterized by multiple time-scales dynamics. In particular, we simplify the DeepESN design by implementing recurrent units with linear activation function, i.e. we introduce linear DeepESN (L-DeepESN). In the analysis of the frequency spectrum of network's states, this approach brings the major advantage of avoiding the effects of harmonic distortion due to non-linear activation functions. To provide a quantitative support to our analysis, we experimentally assess the L-DeepESN model on a variety of progressively more involving versions of the Multiple Superimposed Oscillator (MSO) task [206, 211]. Note that the class of MSO tasks is of particular interest for the aims of this study, especially in light of previous literature results that pointed out the relevant need for multiple time-scales processing ability [100, 168, 211] as well as the potentiality of linear models in achieving excellent predictive results in base settings of the problem [27].

As a further contribution, our work would offer interesting insights on the nature of compositionality in Deep Learning architectures. Typically, deep neural networks consist in a hierarchy of many non-linear hidden layers that enable a distributed

information representation (through learning) where higher layers specialize to progressively more abstract concepts. Removing the characteristic of nonlinearity, and focusing on the ability to develop a hierarchical diversification of temporal features (prior to learning), our analysis sheds new light into the true essence of layering in deep RNN even with *linear* recurrent units.

First, in the next Section 3.4.1 we introduce the L-DeepESN model. Then, in Section 3.4.2 we define the experimental setting. Finally, in Sections 3.4.3 and 3.4.4 we analyze the hierarchical nature of temporal representations in L-DeepESN, presenting the outcomes of the signal processing analysis of the developed system dynamics and the experimental results on the MSO tasks.

3.4.1 Linear Deep Echo State Networks

We define the L-DeepESN architecture as a DeepESN with linear activation functions. Thereby, the state transition function is obtained from Equations 3.1 and 3.2 (In Section 3.2) using the identity function **id** as activation function. By referring to the case of leaky integrator reservoir units [100], and omitting the bias terms for the ease of notation, the state transition function of the first layer is given by the following equation:

$$\mathbf{x}^{(1)}(t) = (1 - a^{(1)})\mathbf{x}^{(1)}(t-1) + a^{(1)}(\mathbf{W}_{\text{in}}\mathbf{u}(t) + \hat{\mathbf{W}}^{(1)}\mathbf{x}^{(1)}(t-1)), \tag{3.17}$$

whereas the state transition of layer $i > 1$ is ruled by the equation:

$$\mathbf{x}^{(i)}(t) = (1 - a^{(i)})\mathbf{x}^{(i)}(t-1) + a^{(i)}(\mathbf{W}^{(i)}\mathbf{x}^{(i-1)}(t) + \hat{\mathbf{W}}^{(i)}\mathbf{x}^{(i)}(t-1)), \tag{3.18}$$

where $a^{(i)} \in [0, 1]$ is the leaking rate parameter at layer i, $\mathbf{W}_{\text{in}} \in \mathbb{R}^{N_R \times N_U}$ is the input weight matrix, $\mathbf{W}^{(i)} \in \mathbb{R}^{N_R \times N_R}$ is the weight matrix of the inter-layer connections from layer $i-1$ to layer i, $\hat{\mathbf{W}}^{(i)} \in \mathbb{R}^{N_R \times N_R}$ is the matrix of recurrent weights of layer i, and $\mathbf{f}$ denotes the element-wise application of the activation function of the recurrent units. A null initial state is considered for the reservoirs in all the layers, i.e. $\mathbf{x}^{(i)}(0) = \mathbf{0}$ for all $i = 1, \ldots, N_L$.

The initialization, the training and the output computation of the network are performed as in DeepESN (see Section 3.2). In particular, the values of matrices $\hat{\mathbf{W}}^{(i)}$, $\mathbf{W}_{\text{in}}$ and $\{\mathbf{W}^{(i)}\}_{i=2}^{N_L}$ are randomly initialized and left untrained. Moreover, as in DeepESN, the only learned parameters of the network are those pertaining to the readout layer.

In this context it also interesting to observe that the use of linearities allows us to express the evolution of the whole system by means of an algebraic expression that describes the dynamics of an equivalent single-layer recurrent system with the same total number of recurrent units. Specifically, given $\mathbf{x}(t) = (\mathbf{x}^{(1)}(t), \mathbf{x}^{(2)}(t), \ldots, \mathbf{x}^{(N_L)}(t)) \in \mathbb{R}^{N_L N_R}$ the global state of the network, the dependence of $\mathbf{x}(t)$ from $\mathbf{x}(t-1)$ can be expressed as:

$$\mathbf{x}(t) = \mathbf{V}\mathbf{x}(t-1) + \mathbf{V}_{\text{in}}\mathbf{u}(t), \tag{3.19}$$

where both $\mathbf{V} \in \mathbb{R}^{N_L N_R \times N_L N_R}$ and $\mathbf{V}_{\text{in}} \in \mathbb{R}^{N_L N_R \times N_U}$ can be viewed as block matrices, with block elements denoted respectively by $\mathbf{V}_{i,j} \in \mathbb{R}^{N_R \times N_R}$ and $\mathbf{V}_{\text{in},i} \in \mathbb{R}^{N_R \times N_U}$, i.e.:

$$\mathbf{x}(t) = \begin{bmatrix} \mathbf{V}_{1,1} & \ldots & \mathbf{V}_{1,N_L} \\ \vdots & \ddots & \vdots \\ \mathbf{V}_{N_L,1} & \ldots & \mathbf{V}_{N_L,N_L} \end{bmatrix} \mathbf{x}(t-1) + \begin{bmatrix} \mathbf{V}_{\text{in},1} \\ \vdots \\ \mathbf{V}_{\text{in},N_L} \end{bmatrix} \mathbf{u}(t). \tag{3.20}$$

Noticeably, the layered organization imposes a lower triangular block matrix structure to $\mathbf{V}$ such that in the linear case its blocks can be computed as:

$$\mathbf{V}_{i,j} = \begin{cases} \mathbf{0} & \text{if } i < j \\ (1 - a^{(i)})\mathbf{I} + a^{(i)}\hat{\mathbf{W}}^{(i)} & \text{if } i = j \\ (\prod_{k=j+1}^{i} a^{(k)}\mathbf{W}^{(k)})\big((1 - a^{(j)})\mathbf{I} + a^{(j)}\hat{\mathbf{W}}^{(j)}\big) & \text{if } i > j. \end{cases} \tag{3.21}$$

Moreover, as concerns the input matrix, we have:

$$\mathbf{V}_{\text{in},i} = \begin{cases} a^{(1)}\mathbf{W}_{\text{in}} & \text{if } i = 1 \\ (\prod_{k=2}^{i} a^{(k)}\mathbf{W}^{(k)})a^{(1)}\mathbf{W}_{\text{in}} & \text{if } i > 1. \end{cases} \tag{3.22}$$

The mathematical description provided here for the L-DeepESN case is particularly helpful in order to highlight the characterization resulting from the layered composition of recurrent units. Indeed, as described in Section 3.2.1 from an architectural perspective, a deep RNN can be seen as obtained by imposing a set of constraints to the architecture of a single-layer fully connected RNN with the same total number of recurrent units (see Figure 3.3). In this respect, the use of linear activation functions has the effect of enhancing the emergence of such constrained characterization and making it visible through the peculiar algebraic organization of the state update as described by Equations 3.20, 3.21 and 3.22. Indeed, the constrained structure given by the layering factor is reflected in the (lower triangular

block) structure of the matrix $\mathbf{V}$ that rules the recurrence of the whole network dynamics in Equation 3.20. In particular, the last line of Equation 3.21 highlights the progressive filtering effect on the state information propagated towards the higher levels in the network, modulated by the leaking rates and through the magnitude of the inter-layer weights values. Similarly, the last line of Equation 3.22 shows the analogous progressive filtering effect operated on the external input information for increasing level's depth.

Thereby, although from the system dynamics viewpoint it is possible to find a shallow recurrent network that is equivalent to an L-DeepESN, the resulting form of the matrices that rules the state evolution, i.e. $\mathbf{V}$ and $\mathbf{V}_{\text{in}}$, has a distinct characterization that is due to the layered construction. Moreover, note that the probability of obtaining such matrices $\mathbf{V}$ and $\mathbf{V}_{\text{in}}$ by means of standard random reservoir initialization is negligible. Noteworthy, the aforementioned architectural constraints imposed by the hierarchical construction are reflected also in the ordered structure of the temporal features represented in higher levels of the recurrent architecture, as investigated for linear reservoirs in Section 3.4.3, and as observed, under a different perspective and using different mathematical tools, in the non-linear case in 3.3.1.

3.4.2 Multiple Superimposed Oscillators Tasks

Here we present the class of MSO tasks used for the experimental assessment of L-DeepESN.

An MSO task consists in a next-step prediction on a 1-dimensional time series, i.e. for each time-step t the target output is given by $y_{\text{target}}(t) = u(t+1)$, where $N_U = N_Y = 1$. The considered time series is given by a sum of sinusoidal functions:

$$u(t) = \sum_{i=1}^{n} sin(\varphi_i t), \tag{3.23}$$

where n denotes the number of sinusoidal functions, φ_i determines the frequency of the i-th sinusoidal function and t is the index of the time-step. In the following, we use the notation MSOn to specify the number n of sinusoidal functions that are accounted in the task definition. The φ_i coefficients in Equation 3.23 are set as in [139, 109], i.e. $\varphi_1 = 0.2, \varphi_2 = 0.331, \varphi_3 = 0.42, \varphi_4 = 0.51, \varphi_5 = 0.63, \varphi_6 = 0.74, \varphi_7 = 0.85, \varphi_8 = 0.97, \varphi_9 = 1.08, \varphi_{10} = 1.19, \varphi_{11} = 1.27, \varphi_{12} = 1.32$. In particular, in our experiments we focus on versions of the MSO task with a number

of sine waves n ranging from 5 to 12. This allows us to exercise the ability of the RC models to develop a hierarchy of temporal representations in challenging cases where the input signal is enriched by the presence of many different time-scales dynamics. Besides, note that summing an increasing number of sine waves with frequencies that are not integer multiples of each other makes the prediction task harder due to the increasing signal period. An example of the input signal for the MSO12 task is given in Figure 3.11. For all the considered settings of the MSO task,

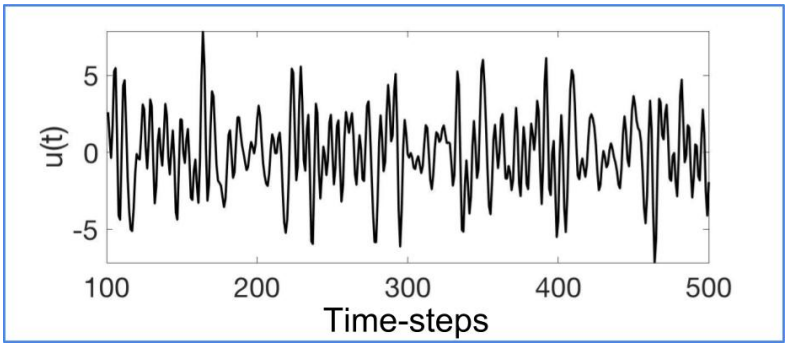


Figure 3.11: A 400 time-steps long excerpt of the input sequence for the MSO12 task.

the first 400 steps are used for training (with a washout of length 100), time-steps from 401 to 700 are used for validation and the remaining steps from 701 to 1000 are used for test.

In our experiments, we used L-DeepESN with N_L levels, each consisting in a fully connected reservoir with N_R units. We assumed that $\mathbf{W}_{\text{in}}$ and $\{\mathbf{W}^{(i)}\}_{i=2}^{N_L}$ are initialized with the same scaling parameter $scale_{\text{in}}$, and we used the same value of the spectral radius and of the leaking rate in every level, i.e. $\rho^{(i)} = \rho$ and $a^{(i)} = a$ for every i. For readout training we used ridge regression. Table 3.7 reports the range of values considered for every hyper-parameter considered in our experiments.

Hyper-parameter	Values considered for model selection
number of levels N_L	10
reservoir size N_R	100
input scaling $scale_{\text{in}}$	0.01, 0.1, 1
leaking rate a	0.1, 0.3, 0.5, 0.7, 0.9, 1.0
spectral radius ρ	0.1, 0.3, 0.5, 0.7, 0.9, 1.0
ridge regression regularization λ_r	$10^{-11}, 10^{-10}, ..., 10^{0}$

Table 3.7: Hyper-parameters values considered for model selection on the MSO tasks.

In order to evaluate the predictive performance on the MSO tasks, we used the

normalized root mean square error (NRMSE), calculated as follows:

$$\mathrm{NRMSE} = \sqrt{(\sum_{\mathrm{t=1}}^{\mathrm{T}}(\mathrm{y_{target}(t)} - \mathrm{y(t)})^2)/(\mathrm{T}\sigma^2_{\mathrm{y_{target}(t)}})}, \tag{3.24}$$

where T denotes the sequence length, $y_{\text{target}}(t)$ and $y(t)$ are the target and the network's output at time t, and $\sigma^2_{y_{\text{target}}(t)}$ is the variance of y_{target}. For each reservoir hyper-parametrization, we independently generated 10 reservoir guesses, the predictive performance in the different cases has been averaged over such guesses and then the model's hyper-parameterization has been selected on the validation set.

In the following Sections 3.4.3 and 3.4.4 we respectively evaluate our approach from a qualitative perspective, by analyzing the frequencies of the state activations developed in the different reservoir levels, and from a quantitative point of view, comparing the predictive performance of L-DeepESN with related literature models.

3.4.3 Multiple Frequency Differentiation

In this section, we investigate the temporal representation developed by the reservoirs levels in an L-DeepESN, using as input signal the sequence considered for the MSO12 task, featured by rich dynamics with known multiple time-scales characterization (see Equation 3.23). We performed the analysis on the selected model according to the experimental settings defined in the previous Section 3.4.2. Thereby, we considered a L-DeepESN with $N_R = 100$, $N_L = 10$, $scale_{\text{in}} = 1$, $a = 0.9$ and $\rho = 0.7$, averaging the results over 100 reservoir guesses. In our analysis, we first computed the states obtained by running the L-DeepESN on the input sequence. Then, we performed the Fast Fourier Transform (FFT) [47] algorithm on the states of all the recurrent units over the time, normalizing the obtained values in order to enable a qualitative comparison. Finally, we averaged the FFT values on a layer-by-layer basis.

Figures 3.12**a**), 3.12**b**), 3.12**c**) and 3.12**d**) show FFT values obtained for progressively higher levels of L-DeepESN which respectively focus on levels 1, 4, 7 and 10. These figures represent the state signal in the frequency domain, where it is possible to see 12 spikes corresponding to the 12 sine waves components of the input[4]. Looking at the magnitude of the FFT components, i.e. at the height of the spikes in plots, we can have an indication of how the signals are elaborated by the individual recurrent levels. We can see that the state of the reservoir at

[4]Note that radians per second φ_i in Equation 3.23 are equivalent to $\frac{\varphi_i}{2\pi}$ Hz (see Figures 3.12).

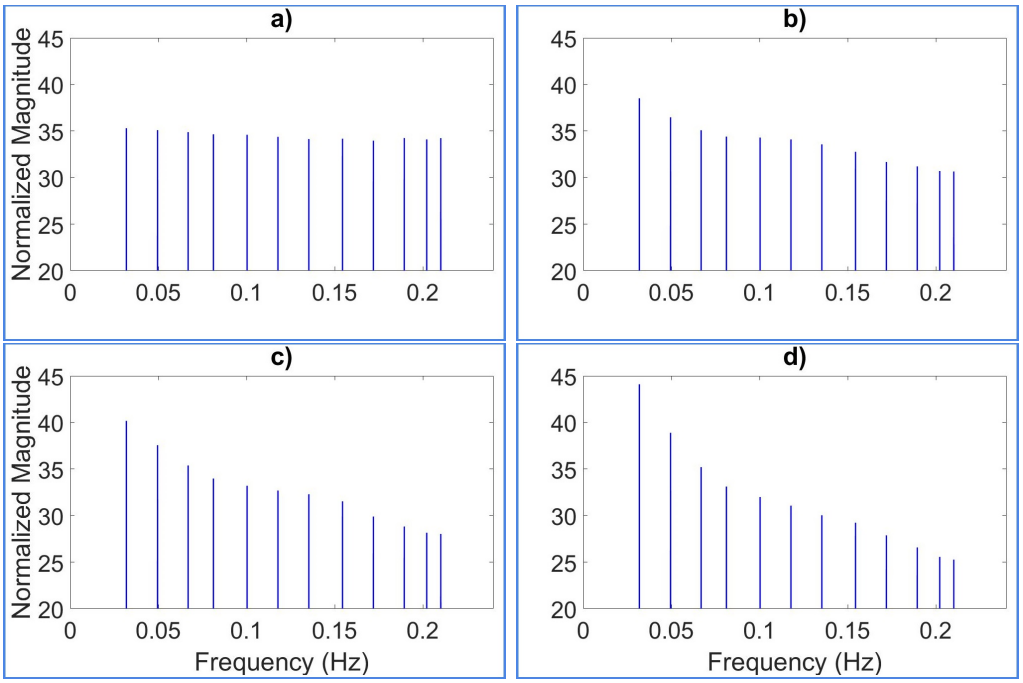


Figure 3.12: FFT components of reservoir states in progressively higher levels of L-DeepESN, **a**) level 1, **b**) level 4, **c**) level 7, **d**) level 10.

level 1 shows FFT components all with approximately the same magnitude. The FFT components of reservoir states at levels 4, 7 and 10, instead, show diversified magnitudes. Specifically, we can see that in higher levels of the network higher frequency components are progressively filtered, and lower frequency components tend to have relative higher magnitudes. This confirms the insights on the progressive filtering effect discussed in Section 3.4.1 in terms of mathematical characterization of the system.

Results in Figure 3.12 show that the hierarchical construction of recurrent models leads, even in the linear case, to a representation of the temporal signal that is sparsely distributed across the network, where different levels tend to focus on different magnitudes of frequency components. Moreover, the higher is the level, the stronger is the focus on lower frequencies, hence the state signals emerging in higher levels are naturally featured by coarser time scales and slower dynamics. Thereby, the layered organization of the recurrent units determines a temporal representation that has an intrinsic hierarchical structure. According to this, the multiple time scales in the network dynamics are ordered depending to the depth of reservoirs' levels. In the next Session 3.4.4, we evaluate the performance obtained by L-DeepESN trained on the MSO tasks exploiting the inherent hierarchical distributed temporal representation developed among the layers as described above.

3.4.4 Predictive Performance

In this section, we compare the quantitative results obtained by the L-DeepESN model with the performance reported in literature on recent (more complex and richer) variants of the MSO task, with a number of sine waves n varying from 5 to 12. Table 3.8 provides a comparison among the NRMSE achieved on the test set by L-DeepESN, neuro-evolution [139], balanced ESN [109], ESN with infinite impulse response units (IIR ESN) [89] and Evolino [168] on the considered MSO tasks. Furthermore, in the same table, we report the performance achieved by linear ESN built with a single fully connected reservoir (L-ESN), considering the same range of hyper parameters and total number of recurrent units as in the L-DeepESN case.

Task	L-DeepESN	L-ESN	n.-evolution [139]	balanced ESN [109]	IIR ESN [89]	Evolino [168]
MSO5	$6.75 \cdot 10^{-13}$	$7.14 \cdot 10^{-10}$	$4.16 \cdot 10^{-10}$	$1.06 \cdot 10^{-6}$	$8 \cdot 10^{-5}$	$1.66 \cdot 10^{-1}$
MSO6	$1.68 \cdot 10^{-12}$	$5.40 \cdot 10^{-9}$	$9.12 \cdot 10^{-9}$	$8.43 \cdot 10^{-5}$	-	-
MSO7	$5.90 \cdot 10^{-12}$	$5.60 \cdot 10^{-8}$	$2.39 \cdot 10^{-8}$	$1.01 \cdot 10^{-4}$	-	-
MSO8	$1.07 \cdot 10^{-11}$	$2.08 \cdot 10^{-7}$	$6.14 \cdot 10^{-8}$	$2.73 \cdot 10^{-4}$	-	-
MSO9	$5.34 \cdot 10^{-11}$	$4.00 \cdot 10^{-7}$	$1.11 \cdot 10^{-7}$	-	-	-
MSO10	$8.22 \cdot 10^{-11}$	$8.21 \cdot 10^{-7}$	$1.12 \cdot 10^{-7}$	-	-	-
MSO11	$4.45 \cdot 10^{-10}$	$1.55 \cdot 10^{-6}$	$1.22 \cdot 10^{-7}$	-	-	-
MSO12	$5.40 \cdot 10^{-10}$	$1.70 \cdot 10^{-6}$	$1.73 \cdot 10^{-7}$	-	-	-

Table 3.8: Test NRMSE obtained by L-DeepESN, L-ESN, neuro-evolution (n.-evolution), balanced ESN, IIR ESN and Evolino on the MSO5-12 tasks.

Noteworthy, the proposed L-DeepESN model outperformed the best literature results of about 3 or 4 orders of magnitude on all the MSO settings. Furthermore, test errors obtained by L-ESN are always within one order of magnitude of difference with respect to the best state-of-the-art results. These aspects confirms the effectiveness of the linear activation function on this task, as also testified by our preliminary results that showed poorer performance for RC networks with *tanh* units, unless forcing the operation of the activation function in the linear region (i.e., considering small inputs weights). Therefore, these results suggest that L-DeepESN can outperforms other non-linear deep architectures such as DeepESN on tasks characterized by linearity (i.e., linear compositions of non-linear functions as in MSO) or on tasks in which the memory capacity is particularly relevant [45].

Moreover, L-DeepESN always performed better then L-ESN. On the basis of the known characterization of the MSO task, our results confirm the quality of the hierarchical structure of recurrent reservoirs in representing multiple time-scales dynamics with respect to its shallow counterpart.

For the sake of completeness, we performed a further comparison considering L-DeepESNs with the same number of total recurrent units used by the other ESN

models taken into account from literature. In particular, balanced ESN used a maximum number of 250 units for model selection on the MSO5, MSO6, MSO7 and MSO8 tasks, while IIR ESN implemented 100 units on the MSO5 task (see results in Table 3.8). L-DeepESN with $N_L = 10$ and $N_R = 25$ (i.e. a total of 250 recurrent units) performed better than balanced ESN, obtaining a test NRMSE of $1.20 \cdot 10^{-11}$, $8.73 \cdot 10^{-11}$, $2.42 \cdot 10^{-10}$ and $9.06 \cdot 10^{-10}$, on the MSO5, MSO6, MSO7 and MSO8 tasks, respectively. Moreover, even L-DeepESN with $N_L = 10$ and $N_R = 10$ (i.e. a total of 100 recurrent units) obtained a better performance than IIR ESN, achieving a test error of $7.41 \cdot 10^{-11}$ on the MSO5 task.

3.5 Discussion

In this Chapter, we proposed the study of deep RC architectures to analyze the inherent characteristics of deep RNNs. In particular, quantitative and qualitative studies, performed on DeepESN model and architectural baselines (described in Section 3.2.1), highlighted a number of major advantages in the use of layering in deep RNN architecture.

The hierarchical organization of the reservoir in successive layers is naturally reflected into the structure of the developed system dynamics. Specifically, it has been experimentally observed in Section 3.3 that the global state of a DeepESN tends to develop a multiple time-scales representation of the input history, hierarchically ordered along the layers of the recurrent architecture. In particular, higher layers showed progressively slower dynamics in the conditions, setup and tasks analyzed in Section 3.3. In this regard, an interesting observation is that the hierarchical structure of DeepESN state representations can be achieved even in the case in which all the reservoir layers share the same values for the hyper parameters. Another relevant outcome of Section 3.3 is that IP adaptation applied in conjunction with a layered recurrent organization is able to further enhance the effect of temporal scales differentiation across the layers.

A hierarchical construction of the reservoir is also beneficial in terms of increasing the richness of the developed state dynamics. This has been experimentally observed in Section 3.3.3, by measuring the averaged entropy of DeepESN states, using the IP rule for unsupervised adaptation and in comparison to the shallow case. Moreover, experimental evidences reported Section 3.3.4 showed that DeepESNs are able to considerably and consistently outperform corresponding shallowESN settings (with the same total number of recurrent units) in terms of MC.

The multiplicity of temporal representations developed by the global internal state of a DeepESN has been also analyzed in Section 3.4 in terms of the frequency spectrum of state components. In particular, considering a deep recurrent architecture with linear activation functions, we defined the mathematical characterization of a dynamical system for describing the progressive filtering computed by a deep RNN on the states of higher layers. Thereby, DeepESN states produce a *multiple frequency representation* of the input information, distributed through layers, where higher layers tended to focus on lower frequencies as illustrated in Section 3.4.3.

3.5.1 Subsequent studies on deep randomized RNNs

Recent works related to the aspects treated in this thesis as well as to deep RC paradigm introduced theoretical studies [55, 67, 66], further investigations on the memory capacity of deep RNNs [49] and real-world applications on ambient assisted living field [56]. In particular, on the theoretical side, studies in the field of dynamical system theory [55] showed that reservoir states in different layers of DeepESN are able to develop dynamics that are qualitatively different in terms of contractive behavior. Specifically, as analyzed in [55] (in a basic setting without IP learning), when the DeepESN is initialized using the same hyper parameters for the scaling of reservoirs matrices in all the levels of the architecture, progressively higher layers tend to be characterized by progressively less contractive dynamics. Furthermore, stability analysis in presence of driving inputs, conducted through the study of Lyapunov exponents in [66, 67], pointed out that layered RNN architectures, compared to shallow counterparts in condition of an equal number of recurrent units, show a dynamical behavior that is naturally pushed closer to the *edge of stability*. This represents a transition condition of states regime near which the RNN system exhibits a rich internal representation of the driving input signals and high performance in tasks requiring long memory spans [123, 16].

3.5.2 Time-scales Differentiation in RC models

The aspect of temporal scales differentiation in RC models has been addressed in literature also from a different, though related, line of architectural studies. These are based on the idea of structuring the reservoir into sub-groups, or sub-reservoirs, characterized by different dynamical properties with the aim to achieve a decoupling among the state dynamics [211], an idea that has been pursued also outside of the RC context e.g. in [212]. A recent development, described in [151], proposed an

incremental approach to the construction of the sub-groups reservoir organization. Differently from the DeepESN model, all of these architectural variants are based on a structured but non-hierarchical organization of the recurrent dynamical part, i.e. they are shallow networks purposely designed to achieve a multiplicity of temporal scales by construction. The experimental comparison between the two approaches, studied in Section 3.3.1, pointed out the actual relevance of a layered architectural construction, in which pools of recurrent units are progressively more distant from the input (and there is no feedback from higher to lower layers). The main aspects regarding DeepESN approaches are also summarized in two recent surveys [53, 58].

3.5.3 Subsequent Studies on Hierarchical RC structures

Further works introduced studies based on different hierarchical RC structures. Empirical studies presented in [127] show the advantages obtained by multi-layered RC architectures on time-series benchmarks in the RC area. Another proposed architecture is characterized by layered reservoirs implemented by means of cellular automata [138, 135]. The use of layering in cellular automata can improve the performance of the model on artificial tasks [138, 135]. Finally, studies proposed in [215] analyze the characteristics of emergent stimulus representations in a hierarchy of recurrent modules of spiking neural networks. Interestingly, the evidences reported in [215] in a context of spiking neural models highlighted that higher layers are characterized by faster dynamics. Also in our studies (presented in the next Chapter 4), we empirically show that it is possible to obtain faster dynamics in the higher layers of DeepESN models by using IP adaptation on complex real-world tasks.

3.6 Conclusions

In this Chapter, we have proposed an experimental analysis of state dynamics in deep RNN architectures, targeted at assessing the real effect of layering on the development of a hierarchical representation of the temporal information. In particular, the recourse to RC networks allowed us to conduct such analysis separately from learning aspects.

Despite the observations that stacking recurrent layers is just an architectural constraint to a fully connected RNN, and that a shallow reservoir already provides a rich pool of varied state dynamics by construction, the experimental evidences

in this Chapter have shown that it is possible to exploit the same factors that influence the dynamics of shallow recurrent architectures to achieve a temporal data representation at multiple levels of abstractions through a layered network organization in deep RNNs.

In particular, the introduction of the DeepESN model allowed us to study the intrinsic properties of deep layered recurrent architectures in terms of time-scales differentiation, and highlight such properties in comparison to purposely introduced baseline models. This allowed us to evaluate the effect of architectural factors such as the progressive distance of higher layers from the external input (versus DeepESN-IA) and the effective hierarchical interplay among layers (versus GroupedESN).

In Section 3.3, experiments on two benchmark datasets have shown the synergy between stacking reservoir layers and the role of already known RC parameters. On the one hand, such analysis provided insights on the amplification of the effect of these RC parameters in a deep architecture. On the other hand, it allowed us to propose effective strategies to enhance the time-scale differentiation among layers using different values of the leaky parameter and of the spectral radius, or by unsupervised IP learning focused only the parameters of the activation function. This allows us to preserve the efficiency of the RC approach, without resorting to a full RNN training (extended to all the units parameters).

More in detail, the variability of parameters of reservoir design ruling the speed of dynamics in response to the input, i.e. the leaky parameter, and the memory length, i.e. the spectral radius, could effectively amplify the emergence of multiple (separated) time scales, hierarchically ordered across the layers of a DeepESN.

The use of an efficient technique for unsupervised adaptation of (only) the parameters of the reservoir activation functions, i.e. IP learning, has shown a great impact on the development of multiple time scales (enhanced in deep models). Such impact has been investigated also in terms of improved richness of reservoir dynamics by measuring the entropy of reservoir state activations, showing that the known effect of IP learning is actually progressively enhanced among the layers of a DeepESN architecture. Furthermore, the advantages brought by the proposed approaches have been shown also on the MC task, showing that DeepESN allows to improve the short term memory capacity with respect to the shallow case, and that the known effect of IP learning on the MC task is greatly exalted by the use of a layered architecture.

Overall, after assessing the intrinsic architectural properties of general deep layered RNN in representing different time-scale dynamics, more interestingly

for RC modeling, the results of our experimental analysis pointed out the actual relevance of the interplay between layering and RC parameters aspects on the diversification of temporal representations. In particular, the proposed approaches allowed us to achieve a time-scale differentiation in deep models that is higher with respect to a standard ESNs without a layered structure, and led to explicitly address the concept of including temporal data representation at different level of abstraction within the RC paradigm.

In Section 3.4, we take a step forward in the study of the temporal features naturally emerging in deep RNNs by means of frequency analysis tools. For this purpose, we have studied the inherent properties of hierarchical linear RNNs by analyzing the frequency of the states signals emerging in the different levels of the recurrent architecture. The use of linear activations allowed us to define an intrinsic mathematical characterization of layered recurrent architectures and to perform a frequency analysis without harmonic distortion. The FFT analysis revealed that the stacked composition of reservoirs in a L-DeepESN tends to develop a structured representation of the temporal information. Exploiting an incremental filtering effect, states in higher levels of the hierarchy are biased towards slower components of the frequency spectrum, resulting in progressively slower temporal dynamics. In this sense, the emerging structure of L-DeepESN states can be seen as an echo of the multiple time scales present in the input signal, distributed across the layers of the network. The hierarchical representation of temporal features in L-DeepESN has been exploited to address recent challenging versions of the MSO task. Experimental results showed that the proposed approach dramatically outperforms the state of the art on the MSO tasks, emphasizing the relevance of the hierarchical temporal representation and also confirming the effectiveness of linear signal processing on the MSO problem.

Overall, the analysis proposed in this Chapter paves the way to further studies on the design of novel deep RNN models for efficient representation learning on sequences. Future developments deserve to move from the current insights to the design and concrete set up of new learning models boosted by an enriched representation of the input dynamics, exploiting the time-scale differentiation developed through the layers to solve complex tasks that require/involve processing time-series data at different levels of time granularity. The opening of this line of research would contribute to achieving new findings that are demanded to result in a relevant breakthrough in the area of efficiently learning from sequential and temporal data.

Moreover, we showed a concrete evidence that layering is an aspect of the network construction that is intrinsically able to provide a distributed and hierarchical feature representation of temporal data. Our analysis pointed out that this is possible even without (or prior to) learning of the recurrent connections, and releasing the requirement for nonlinearity of the activation functions.

Finally, the analysis tools developed in this Chapter that allowed us to study the intrinsic role of layering in deep RNN pave the way to define design approaches based on qualitative and quantitative measures. In particular, in the next Chapter, we introduce a design approach for efficient deep RNN architectures based on measures computed by frequency analysis tools.

Analysis and Design of Deep RNNs

The content presented in this Chapter is based on work published in [62] (see Section 1.5 for details).

In previous Chapter we developed analysis tools for the study of intrinsic characterizations of deep RNNs (under different aspects). Moreover, the DeepESN model (introduced in Section 3.2) resulted an efficient deep recurrent model able to develop rich temporal representations of input signals characterized by multiple time-scales dynamics. On such considerations, in this Chapter we aim to introduce a novel approach for the efficient design of deep RNN based on DeepESN models.

The Chapter is organized as follows. In Section 4.1, we present main aspects of the design approach that we aim to introduce. Then, in Section 4.2 we define the proposed method for designing DeepESN architectures, providing an analysis of the involved advantages in terms of computational cost and analyzing it under an ad hoc controlled scenario. We discuss the outcomes of our analysis in Section 4.3, and we present conclusions in Section 4.4. For the sake of presentation, the experimental evaluations of this approach on real-world applications are presented in the next Chapter (in Sections 5.3.1 and 5.3.2).

4.1 Introduction

The analysis conducted in the previous Chapter highlighted the potential advantages of layering as a factor of architectural design in the development of a multiple time-scales dynamical behavior. Starting from this intrinsic characterization, we can thus ask whether the number of layers in the architecture is actually providing a sufficiently diversified behavior, and, on the other hand, whether adding new layers is still effective in terms of dynamical differentiation or not. In other words, in this chapter we tackle the problem of how to choose the number of layers in a deep

recurrent architecture, which currently represents one of the main open questions in the deep learning area. Differently from the work in [145], which describes different possible ways of introducing deepness into the architecture of an RNN trained with stochastic gradient descent, here we explicitly address the issue of how to operatively set the number of layers in deep stacked recurrent models, based on the properties of the specific driving input signals and without the training of recurrent units.

Specifically, in this Chapter we propose an automatic method for the design of deep RNNs, based on frequency analysis and aimed at appropriately exploiting the differentiation of temporal representation in DeepESN architectures. The hypotheses (that delineate the scope of the proposed approach) are that the input signals are multi-scale and that the differences in the time scales are important for the learning task at hand. Given these hypotheses, we aim at exploiting such differences, tailoring the layered architecture to the characteristics of the input signals, by adding layers only as long as the changes in the frequency spectrum are effective through layering. This will be crucial for the final classification/regression performance, under the assumed conditions, proportionally to the effectiveness of the readout training in grasping/modulating the layered differentiation. Under such conditions, the proposed approach has the advantage to determine the proper number of recurrent layers avoiding to apply the training algorithm for each possible number of recurrent layers explored by the usual trial and error approach.

Besides, a secondary objective is also to bring attention to a more general methodological aspect concerning the analysis of multi-layered recurrent architectures by means of signal processing tools for investigation aims, with a focus on monitoring the filtering effect on input signals through the recurrent layers. This aspect is concretely exploited for design purpose in this work, providing an unsupervised approach to determine the number of layers for a deep recurrent architecture on the basis of the data at hand, while conserving a more general flavor for future research.

Based on the analysis of quantitative measures of frequency spectrum in the state space, we define an iterative procedure to assess the diversification of multiple time-scales dynamics among layers. In this Chapter, we analyze and refine our design method on a controlled scenario characterized by signals with multiple time-scales dynamics, studying qualitative and quantitative aspects of frequency analysis in layers states. Subsequently, these analysis (and results) are exploited in the next Chapter (in Sections 5.3.1 and 5.3.2), where we experimentally evaluate

the method on challenging real-world tasks in the area of temporal processing of time series featured by multiple times-scales, namely music processing and speech processing.

4.2 Spectral Analysis and Depth of DeepESN

In the following sections, we introduce our approach based on the spectral analysis for automatically determine the depth of DeepESN architectures. First, in Section 4.2.1 we define the iterative algorithm, then in Section 4.2.2 we discuss its advantages with respect to a standard cross-validation approach in terms of computational costs. In Section 4.2.3 we assess our method on a controlled scenario. Finally, in Section 4.2.4 we experimentally analyze the role of the depth in deep RNNs using spectral analysis.

4.2.1 Method

In this section, we aim to define an automatic algorithm, based on spectral analysis, to determine the depth of DeepESN architectures in which every layer encodes a different range of time-scales dynamics. For this purpose, we take inspiration from research findings presented in Sections 3.3 and 3.4 that showed the intrinsic ability of a stack of recurrent layers to develop a multiple time-scales differentiation among the layers even considering the same value of the leaky rate for each recurrent layer and unit. Since in Section 3.4 the aim was the qualitative study of the spectral properties of the state developed through the layers, for those assessments, we considered recurrent layers with linear activation functions in order to avoid harmonic distortion and then to highlight in a clear way the time scales involved in the temporal processing. Differently from Section 3.4, here we consider the non-linear DeepESN model with IP training defined in Section 3.2 because, in this context, we need nonlinearity to compete with the state of the art on real-world tasks. These considerations allowed us to define a simple design method based on building blocks (recurrent layers with same hyper parameters) used to build up the deep architecture. As it is known, a recurrent layer can be studied as a filter [100, 120, 89, 209]. However, differently from filter design, in which the purpose is to design a filter with a specific cut-off frequency, in our work we aim to exploit the richness of dynamics represented in the state of each recurrent layer through the training of the readout component. In such a way, the output layer can adaptively

choose a proper modulation of time scales on the basis of the characteristics of the supervised task to solve. Therefore, the main idea behind the proposed automatic method for network design is to stop adding new layers whenever the filtering effects become negligible, i.e. when adding new layers essentially does not enrich anymore the multiplicity of temporal dynamics developed by the reservoir states.

In order to determine when the filtering effect becomes negligible, we perform a spectral analysis by computing the spectral centroid and the spectral spread (defined below in Equations 4.2 and 4.3) on the state signal of layers. Intuitively, they represent weighted average and bandwidth of frequency spectrum values computed on the state of recurrent layers over time. Spectral centroid tends to converge to a certain value as we add recurrent layers (further details in Section 4.2.4). Therefore, we define a *stop condition* of the iterative algorithm to detect when the shift of spectral centroid converges:

$$|\mu^{(l)} - \mu^{(l-1)}| \leq \sigma^{(l-1)}\eta \tag{4.1}$$

where $0 < \eta < 1$ (see Section 4.2.4 for details regarding the η value). $\mu^{(l)}$ and $\mu^{(l-1)}$ are the spectral centroid computed on state of layers l and $l-1$ respectively, and $\sigma^{(l-1)}$ is the spectral spread computed on the state of layer $l-1$. On the right-hand side of Equation 4.1, the η value is multiplied by $\sigma^{(l-1)}$ in order to take into account also the bandwidth of the spectrum.

In formulas, the spectral centroid (Equation 4.2) and the spectral spread (Equation 4.3) are defined as follows:

$$\mu^{(l)} = (\sum_{j=1}^{k} p_j^{(l)} f_j^{(l)}) / \sum_{j=1}^{k} p_j^{(l)}, \tag{4.2}$$

$$\sigma^{(l)} = \sqrt{(\sum_{j=1}^{k} p_j^{(l)} (f_j^{(l)} - \mu^{(l)})^2) / \sum_{j=1}^{k} p_j^{(l)}}, \tag{4.3}$$

where $\mathbf{f}^{(l)} = [f_1^{(l)}, \ldots, f_k^{(l)}]$ and $\mathbf{p}^{(l)} = [p_1^{(l)}, \ldots, p_k^{(l)}]$ respectively denote the normalized frequencies and the corresponding magnitudes of components, computed over time on the state of layer l, i.e. $\mathbf{x}^{(l)}$, by the FFT algorithm, whereas k is the number of frequency components (i.e., the length of both vectors $\mathbf{p}^{(l)}$ and $\mathbf{f}^{(l)}$). More in detail, the steps performed for the computation of $\mathbf{f}^{(l)}$ and $\mathbf{p}^{(l)}$ are presented in Algorithm 3.

Algorithm 3 FFT of layer state signal

1: **procedure** FFT($\mathbf{x}^{(l)}$)
2: `comps_g = []` ▷ frequency components on guesses
3: **for** `guess in 1, ..., number_of_guesses` **do**
4: `comps_u = []` ▷ frequency components on units
5: **for** `unit in 1, ..., number_of_units` **do**
6: `signal` $= \mathbf{x}^{(l)}$`(unit,:)` ▷ state signal of the unit
7: `timesteps` $=$ `length(signal)`
8: `comps` $=$ `fft(signal)` ▷ Matlab fft function
9: `comps_u(unit,:)` $=$ `abs(comps(`$1 : \lfloor$`timesteps`$/2) \rfloor$`)` ▷ positive fqs
10: `comps_g(guess,:)` $=$ `mean(comps_u)` ▷ average on units (rows)
11: $\mathbf{p}^{(l)} =$ `mean(comps_g)` ▷ average on guesses (rows)
12: $\mathbf{f}^{(l)} = [1 : \lfloor$`timesteps`$/2 \rfloor]/$`timesteps` ▷ normalized frequency (cyc/s)
13: **return** $\mathbf{p}^{(l)}, \mathbf{f}^{(l)}$

Algorithm 3 computes the frequency components for each unit of each reservoir guess. For each layer l, terms $\mathbf{p}^{(l)}$ are obtained by averaging over the reservoir units and guesses. Depending on the number of time-steps, terms $\mathbf{f}^{(l)}$ are computed in order to have normalized frequency components measured as cycles/seconds.

Here we assume to consider a standard model selection process in which for each hyper-parameterization a certain number of reservoir guesses are instantiated (with different random initialization). Given a configuration of hyper parameters of the model, denoted by θ, the design Algorithm 4 selects a number of layers for the network's architecture. The function *computeState*() called inside Algorithm 4, is composed by two steps, first, the IP Adaptation is performed (see Equation 2.35) over the layers and, second, the state of the network is computed and returned. Finally, the number of layers is calculated before training the readout, incrementally considering new layers of recurrent units in the architecture until the stop condition in line 8 of Algorithm 4 (i.e., Equation 4.1) is satisfied or the max number of layers (i.e., M_L) is reached.

4.2.2 Analysis of the Computational Cost

Typically, the number of layers in a deep (RNN) architecture is selected on a validation set through a cross-validation approach that results in an extremely expensive procedure from the computational point of view [3]. The aim of the analysis provided in this sub-section is to quantify the advantage, in terms of

Algorithm 4 Design of DeepESN

procedure DESIGNDEEPESN(θ)
  for `l in 1, ..., `M_L`-1` **do**
    $\mathbf{x}^{(l)} = \texttt{computeState}(l, \theta(l))$ ▷ state on layer l with h-params $\theta(l)$
    $\mathbf{p}^{(l)}, \mathbf{f}^{(l)} = \texttt{FFT}(\mathbf{x}^{(l)})$ ▷ output of Algorithm 3
    $\mu^{(l)} = \frac{\sum_{j=1}^{k} p_j^{(l)} f_j^{(l)}}{\sum_{j=1}^{k} p_j^{(l)}}$
    $\sigma^{(l)} = \sqrt{\frac{\sum_{j=1}^{k} p_j^{(l)} (f_j^{(l)} - \mu^{(l)})^2}{\sum_{j=1}^{k} p_j^{(l)}}}$
    if `l` > 1 **then**
      if $|\mu^{(l)} - \mu^{(l-1)}| <= \sigma^{(l-1)}\eta$ **then**
        return `l`
  return M_L

Algorithm 5 Basic Cross-Validation

for `f in 1, ...,`N_f **do**
  for θ `in 1, ...,`N_θ **do**
    for `l in 1, ...,`M_L **do**
      TRAINRNN(θ,`l`, `f`) ▷ TRAINREADOUT(θ,`l`, `f`) in DeepESN cases
      VALIDATE(θ,`l`, `f`)
return SELECTMODEL(.) ▷ θ obtaining the best result on the validation set

computational cost, of the use of the proposed design algorithm for deep recurrent models with respect to a basic cross-validation approach.

In the case of deep recurrent models, a typical systematic procedure to choose the number of layers consists in training each hyper-parameterization of the network on a training set for every number of layers considered, i.e. using networks with a number of layers from 1 to a maximum number of layers M_L. The number of layers is then chosen (along with the other hyper parameters) to maximize the performance achieved on a validation set. The basic cross-fold validation procedure, considering a number of N_f folds and N_θ hyper-parameterizations, is summarized in Algorithm 5.

For each hyper-parameterization θ and fold `f`, the cost of such procedure is given by:

$$\mathcal{C}_{\text{deep}}(M_L) = \sum_{N_L=1}^{M_L} \mathcal{C}_{\text{tr}}(N_L), \tag{4.4}$$

where $\mathcal{C}_{\text{tr}}(N_L)$ is the cost of training a deep recurrent architecture with N_L recurrent layers.

Algorithm 6 Cross-Validation with the Design Algorithm

1: **for** `f in 1, ...,`N_f **do**
2: **for** θ `in 1, ...,`N_θ **do**
3: `l =` DESIGNDEEPESN(θ, `f`) ▷ output of Algorithm 4
4: TRAINREADOUT(θ,`l`, `f`)
5: VALIDATE(θ,`l`, `f`)
6: **return** SELECTMODEL(.) ▷ θ obtaining the best result on the validation set

Within the context of the deep RC paradigm, the readout layer is the only part of the network that is trained. Accordingly, for DeepESNs the training cost in Algorithm 5, i.e. the $\mathcal{C}_{\text{tr}}$ term in Equation 4.4, is determined by the cost of training the readout. Considering a DeepESN with N_L layers, a training set with N_T time-steps, and adopting a typical direct method based on SVD for choosing the readout's weights, the training cost is given by:

$$\mathcal{C}_{\text{tr\_rc}}(N_L) = \mathcal{O}((N_R N_L)^2 N_T). \tag{4.5}$$

Therefore, using the right-hand side of Equation 4.5 as training cost in Equation 4.4, we can see that the process of choosing the number of layers (for each hyper-parameterization θ and fold `f`) using the basic procedure described by Algorithm 5 is given by:

$$\mathcal{C}_{\text{basic}}(M_L) = \sum_{N_L=1}^{M_L} \mathcal{O}((N_R N_L)^2 N_T) = \mathcal{O}(N_R^2 M_L^3 N_T). \tag{4.6}$$

Compared to the basic procedure explained above, the design methodology proposed here allows to restrict the number of cases for which training is applied, taking into account only the number of layers that are selected by the design Algorithm 4. The resulting selection process is illustrated in Algorithm 6.

We can note that the cost of Algorithm 4 is dominated by the cost of performing the FFT, given by:

$$\mathcal{C}_{\text{fft}}(M_L) = \mathcal{O}(N_R\, M_L\, N_V \log(N_V)), \tag{4.7}$$

where N_V is the size of the validation set. Overall, the cost entailed by the proposed Algorithm 6 is determined by the sum between $\mathcal{C}_{\text{fft}}(M_L)$ in Equation 4.7, which is linear in the total number of recurrent units $N_R\, M_L$, and the cost $\mathcal{C}_{\text{tr\_rc}}(M_L)$ in Equation 4.5, which is quadratic in $N_R M_L$. Accordingly, reducing the total cost to the sole cost of the dominant operation (i.e., training the readout), the cost of

Algorithm 6 (for each hyper-parameterization θ and fold `f`) can be expressed as follows:

$$\mathcal{C}_{\text{design}}(M_L) = \mathcal{O}(N_R^2 M_L^2 N_T). \tag{4.8}$$

Overall, comparing $\mathcal{C}_{\text{design}}$ and $\mathcal{C}_{\text{basic}}$ emerges that the cross-validation procedure that makes use of our proposed design algorithm leads to a clear reduction of the cost (per fold and per hyper-parameterization) that scales with the number of layers M_L. Notice that these benefits are even more enhanced when the number of hyper-parameterizations (N_θ) and folds (N_f) is considered in the total cost of the selection procedure. In such case, the reduction of the cost amounts to $\mathcal{O}(N_\theta N_f M_L)$, which could be quite high both considering deep networks and the typical high dimension of the hyper-parameter space in the RC applications.

4.2.3 Design Experiments in a Controlled Scenario

In this section, we define an artificial task called Frequency Based Classification (FBC) characterized by signals with multiple time-scales dynamics in order to analyze and refine our method on a controlled scenario. Accordingly, we consider an input sequence $\mathbf{s}$ formed by a random concatenation of elements that belong to subsequences $\mathbf{s}_1$ or $\mathbf{s}_0$. The subsequence $\mathbf{s}_1$ contains a sum of impulse trains with periods from 3 to 29, while $\mathbf{s}_0$ contains trains with periods from 3 to 31. The classification task consists in determining, at each time-step t, if the element $\mathbf{s}(t)$ is equal to $\mathbf{s}_1(t)$ or $\mathbf{s}_0(t)$. The FBC dataset is made publicly available for download[1].

In formulas, let $\mathbf{signal}_i$ be an impulse train with period i such that the element t of the signal (i.e., $\mathbf{signal}_i(t)$) is 1 every period i and 0 otherwise. The sequence $\mathbf{s}$ is defined as follows:

$$\mathbf{s} = [\mathbf{s}_0(0), \mathbf{s}_{j_1}(1), ..., \mathbf{s}_{j_n}(n)], \tag{4.9}$$

where $\mathbf{s}_{j_t} \in \{\mathbf{s}_1, \mathbf{s}_0\}$, $\mathbf{s}_1 = \sum_{i=3}^{29} \mathbf{signal}_i$ and $\mathbf{s}_0 = \sum_{i=3}^{31}, \mathbf{signal}_i$. Moreover, the index $j_t \in \{0, 1\}$ is different from $j_{t-1} \in \{0, 1\}$ with probability of 0.01, in formulas:

$$j_{t+1} = \begin{cases} j_t + 1 \ mod \ 2 & \text{with probability } 0.01 \\ j_t & \text{otherwise.} \end{cases} \tag{4.10}$$

Figure 4.1 shows an excerpt of the sequence $\mathbf{s}$. The continuous lines and the dashed lines indicate that the element $\mathbf{s}(t)$ belongs to $\mathbf{s}_1$ or $\mathbf{s}_0$, respectively. Consider that the probability (defined in Equation 4.10) to have a switch between subsequences

[1] <http://www.di.unipi.it/groups/ciml/Data/fbc.html>

$\mathbf{s}_1$ and $\mathbf{s}_0$ is low, for instance in Figure 4.1, this happens only in time-steps, 1107, 1122 and 1194. Therefore, sequence $\mathbf{s}$ tends to have long ranges with elements of the same subsequence $\mathbf{s}_i$. In particular, in the generated sequence $\mathbf{s}$ for this task, the average number of time-steps of such ranges is 88.2. Note that the information involved in a single element $\mathbf{s}(t)$ is not sufficient to discriminate the elements $\mathbf{s}_0(t)$ and $\mathbf{s}_1(t)$. Therefore, the capacity of the model in representing temporal dynamics of the past of $\mathbf{s}(t)$ is relevant to perform the correct classification.

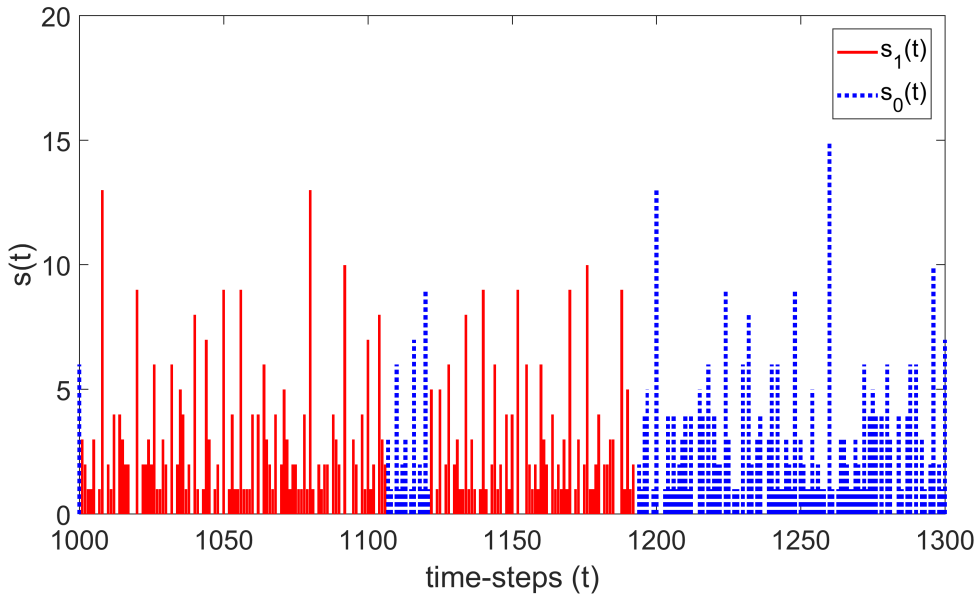


Figure 4.1: A 300 time-step long excerpt of the sequence $\mathbf{s}$ for the FBC task.

The dataset contains a total number of 6000 time-steps, and it is split such that the first 2000 time-steps are used for training, the following 2000 for validation and the last 2000 for test, while the first 20 time-steps of the training set are used for the washout phase. The $\mathbf{y}_{\text{target}}(t)$ target, contained in the labeled dataset $\{\mathbf{y}_{\text{target}}(t), \mathbf{s}(t)\}_{t=1}^{6000}$, is defined as follows:

$$\mathbf{y}_{\text{target}}(t) = \begin{cases} [0\,1]^T & \text{if } \mathbf{s}(t) = \mathbf{s}_1(t) \\ [1\,0]^T & \text{if } \mathbf{s}(t) = \mathbf{s}_0(t). \end{cases}$$

The performance on the FBC task was evaluated in terms of classification accuracy (ACC), i.e., the percentage of correctly classified elements of the sequence $\mathbf{s}$. We applied the design algorithm proposed in Section 4.2.1 to determine the number of DeepESN recurrent layers, considering the network's hyper-parameterizations as specified by the ranges reported in Table 4.1. We fixed the same number

of recurrent units (i.e., $N_R = 100$) in each layer in order to have a comparable frequency spectrum of the temporal dynamics among layers' states. Moreover, the recurrent layers considered have a small/medium number of units because our aim is to consider them as a sort of building blocks for the DeepESN architecture. Finally, we performed model selection on the validation set, using for each hyper-parameterization the number of layers computed by the design algorithm. For each reservoir hyper-parameterization, we independently generated 10 reservoir guesses, and the predictive performance in the different cases has been averaged over such guesses. Moreover, the proposed approach was compared with a shallowESN (a DeepESN model with one recurrent layer). Each model is individually optimized with grid search on hyper-parameters values as specified in Table 4.1, and on a range of total recurrent units in $\{100, 200, ..., 2000\}$. We adopted an experimental setting in which the hyper-parameter values are the same for all layers, in particular, the leaky parameter $a^{(l)}$ and the spectral radius $\rho^{(l)}$ (see Equations 3.1 and 3.2 in Section 3.2) are the same for every layer l. Note that the models are compared on a wide range of hyper parameters: this range is large respect to the standard ranges used in literature for ESN and it is suitable to optimize both shallow and deep configuration without specific bias. The training of free parameters has been performed by means of ridge-regression with regularization term λ_r. For determining the depth of the DeepESN we used Algorithm 4 with $\eta = 0.01$ The choice of such η value is motivated and discussed in Section 4.2.4.

Hyper-parameter	
readout regularization λ_r	$0, 10^{-15}, 10^{-14}, ..., 10^{0}$
input and inter-layer scaling $scale_{\text{in}}$	0.01, 0.1, 1, 10
leaking rate a	0.1, 0.3, 0.5, 0.7, 0.9, 1.0
spectral radius ρ	0.1, 0.3, 0.5, 0.7, 0.9, 1.0
IP standard deviation σ_{IP}	0.1, 1

Table 4.1: Range of DeepESN and shallowESN hyper-parameters values for model selection in the considered tasks.

Table 4.2 shows the training, validation and test classification accuracy achieved by the DeepESN model with the optimal number of recurrent layers obtained by the design algorithm, compared with the accuracy achieved by the shallowESN model.

Model	TR ACC	VL ACC	TS ACC
DeepESN	**83.37 (0.0019) %**	**83.00 (0.0017) %**	**81.77 (0.0056) %**
shallowESN	82.22 (0.0065) %	77.37 (0.0196) %	79.31 (0.0067) %

Table 4.2: Training (TR), validation (VL) and test (TS) classification accuracy (ACC) obtained by DeepESN and shallowESN on the FBC task.

The hyper parameters that obtained the best performance on the validation set are $N_L = 7$ (by the design algorithm), $N_R = 100$, $\rho = 0.9$, $scale_{\text{in}} = 0.1$, $a = 0.1$, $\sigma_{\text{IP}} = 0.01$ and $\lambda_r = 10^{-10}$ for DeepESN and $N_L = 1$, $N_R = 300$, $\rho = 0.9$, $scale_{\text{in}} = 1$, $a = 0.1$, $\sigma_{\text{IP}} = 0.01$ and $\lambda_r = 10^{-7}$ for shallowESN. Results in Table 4.2 show that DeepESN outperforms shallowESN on both validation and test sets, with an accuracy improvement of 5.63% and 2.46%, respectively. It is worth to note that, noticeably, shallowESN obtains worse results on validation and test sets than DeepESN despite the difference between the two models is due to the number of layers in which the recurrent units are arranged, in condition of equal range of possible values of hyper parameters for model selection. This is an instance of the fact that choosing the proper number of recurrent layers in a hierarchical architecture can play a big role in the representation of multiple time-scales dynamics involved in complex signals, with an effect also on the accuracy.

In order to evaluate the quality of the proposed design algorithm in the selection of the number of layers, we compared the performance obtained by our approach with the results achieved using a DeepESN with a number of layers that goes from 1 to 20. Figure 4.2 shows the classification errors ($= 100 - \text{ACC } \%$) obtained on the validation set by DeepESN considering a progressively larger number of layers. In this case, for the sake of analysis, the readout is trained for each configuration to show the comparison.

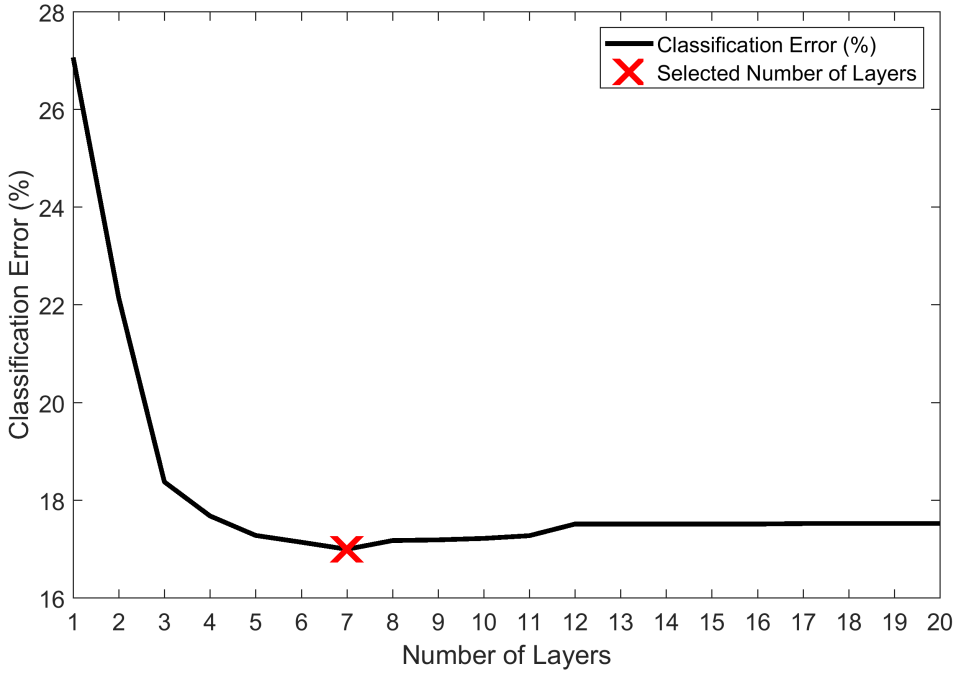


Figure 4.2: Classification error obtained on the validation set of the FBC task by DeepESN architectures with a number of recurrent layers up to 20. Results are obtained through model selection individually performed for each number of layers. The marker 'X' indicates the number of recurrent layers selected by the design algorithm.

In Figure 4.2, the marker 'X' represents the number of layers selected by the design algorithm (7 in our case-study). Noteworthy, the design method added the optimal number of layers in the hierarchy, besided filtering aspects, also with respect to the final accuracy of the global model, i.e. allowing to obtain the lowest error among the considered configurations. In the studied case, these results show that the developed approach allowed us to accurately select the number of layers by just analyzing the time-scales diversification among the temporal dynamics developed in the hierarchy, avoiding to perform the training algorithm for each possible number of recurrent layers.

The qualitative aspects of the temporal representation encoded in the state dynamics of the recurrent layers is further investigated by means of frequency analysis. Figures 4.3a), 4.3b), 4.3c) and 4.3d) show the frequency components computed over time on the state of layers 1, 3, 5 and 7 respectively.

Note that the frequency components of the impulse trains $(1/3, 1/4, 1/5, ..., 1/31)$ that have smaller frequencies have smaller magnitudes. In this task, the frequencies that discriminate $\mathbf{s}_0$ from $\mathbf{s}_1$ are 1/31 and 1/30, represented by the azure and red vertical lines on the left of Figures 4.3a), 4.3b), 4.3c) and 4.3d). In Figure 4.3a),

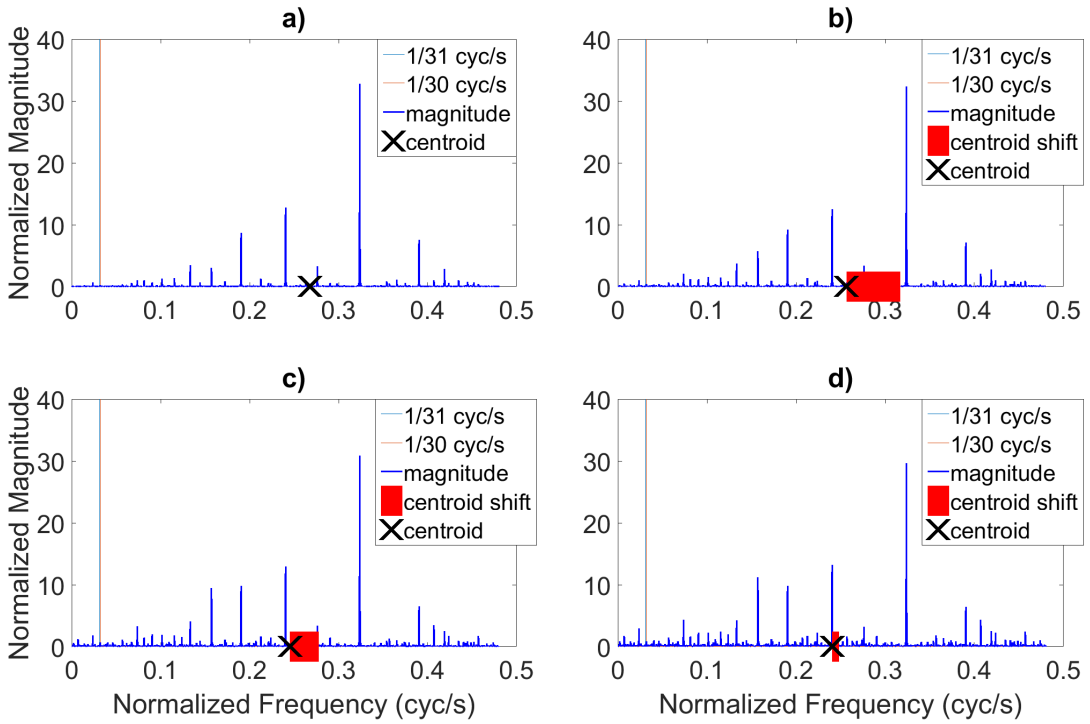


Figure 4.3: Frequency components computed over time on reservoir states, encoding the sequence defined in Equation 4.9, in progressively higher layers of DeepESN. **a**) layer 1, **b**) layer 3, **c**) layer 5, **d**) layer 7. The red range represents the shift of spectral centroid between current and previous layer, multiplied by a factor of 10. Normalized Frequency is expressed in cycles per second (cyc/s).

the components around frequencies 1/31 and 1/30 have almost null magnitudes. Therefore, it is more difficult to perform the classification task by a 1-layered model, as indeed showed by the lower performance achieved by shallowESN on the task. In higher layers, instead, the frequency components around 1/31 and 1/30 are progressively more visible, showing the filtering effect that naturally results in the state computation taking place in successive layers of the architecture. The red range in 4.3b), 4.3c) and 4.3d) represents the difference of frequency mean (i.e., the shift of the spectral centroid) between the current layer and the previous layer (the measure defined on the left side of Equation 4.1) amplified by a factor of 10 in order to make it visually more clear. We can see that the frequency mean, represented by the marker 'X' in 4.3b), 4.3c) and 4.3d), shifts progressively to the left in the higher layers towards low-frequency components. At layer 7 the condition in Equation 4.1 is satisfied and the design algorithm stops adding new recurrent layers. In this regard, it is also worth to note that, in the case illustrated in Figure 4.3, the magnitude of the frequency components, e.g. as measured in a range of 0.01 cyc/s around the vertical line indicated in the plots in correspondence of the frequency 1/31, in layer 7 are quantitatively amplified by more than 500% in comparison to layer 1. Given the quantitative and the qualitative results described so far, we can say that differently from the case of shallowESN, in the higher layers of the DeepESN reservoir hierarchy, the information of the frequency components around 1/31 and 1/30 becomes more easily accessible and can be exploited in order to improve the classification accuracy. Overall, in the analyzed scenario, the proposed design algorithm allows to choose a proper number of recurrent layers, ensuring a rich representation of the input history and at the same time guaranteeing the time-scale differentiation in the hierarchy avoiding to consider further recurrent layers with similar dynamics.

Finally note that, although in Figure 4.2 we show all the layers for the sake of the analysis, the practical aim of the proposed design approach is to stop the algorithm when the condition defined in Equation 4.1 is satisfied in order also to avoid the computation of the FFT in the remaining layers. The studies regarding the existence of other possible optimal points depending on the dataset are postponed to future works.

4.2.4 Experimental Analysis of Depth in the Controlled Scenario

In this section, we empirically evaluate how the spectral centroid (see Equation 4.2), computed by Algorithm 3 over the layers, varies with the depth of the stacked recurrent architecture. This allows us to assess the considered threshold η and to evaluate the effectiveness of the Algorithm 4 and the stop condition defined in Equation 4.1.

A recurrent layer can be studied as a (low/high/band-pass) filter [89, 100, 120, 209]. The effect of the filter determines a cut-off frequency and a roll-off, which tells how the frequency magnitude decreases for increasing (resp. decreasing) frequencies after (resp. before) the cut-off frequency in a low (resp. high) pass filter. Hence, a deep RNN architecture, such is DeepESN, operates as a stack of progressively applied filters. Stacking progressively more filters, i.e. adding layers to a DeepESN, leads the roll-off value to converge towards the cut-off frequency [94], which entails that the spectral centroid converges to a certain asymptotic value. Hence, when a new layer is added, its filtering effect in terms of shifting the spectral centroid of state signals progressively decreases.

In this context, the purpose of the proposed automatic method for network design is to stop adding new layers whenever the filtering effects become negligible, i.e. when adding new layers essentially does not enrich anymore the multiplicity of temporal dynamics developed by the reservoir states. Here, the role of the η parameter in Equation 4.1 is to practically terminate the process of adding layers when the spectral centroid is close enough to its convergence point. Since the convergence can be asymptotic or some numerical errors can lead to small fluctuations in the shift of the spectral centroid computed through layers, the η value should not be 0. However, in order to reach a point that is close enough to the convergence, the η value should be sufficiently small. Overall, the choice of η stems from a reasonable trade-off between such conditions (as typical in any iterative algorithm with asymptotic behaviour, which does not lose generality with a termination cut-off condition). Empirically, we observed that in all considered tasks a value of $\eta = 0.01$ meets this trade-off, being large enough to avoid chasing the asymptote (or following the small fluctuations), and at the same time being sufficiently small to reach a point near the convergence. In order to empirically verify the soundness of this choice, we conducted several experimental evaluations on the FBC task, in this section, and on the real-world tasks (described in the next

Chapter in Sections 5.3.1 and 5.3.2).

Figure 4.4 shows the trend of the spectral centroid obtained from the state of each recurrent layer of the DeepESN, optimized on the FBC task and considering the value of $\eta = 0.01$. The red vertical line represents the number of selected layers.

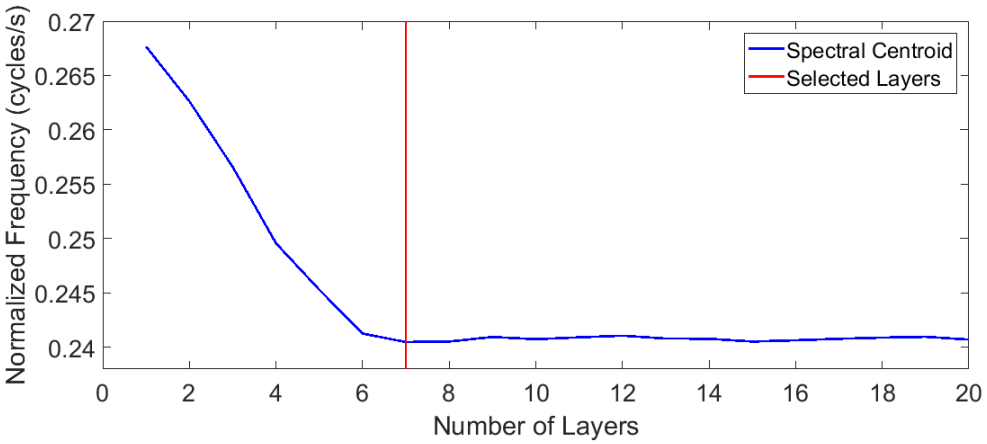


Figure 4.4: Spectral centroid computed on the state of DeepESN layers optimized on FBC task. The red vertical line indicates the number of layers selected by the design algorithm. Threshold η value is set to 0.01. Normalized Frequency is expressed in cycles per second (cyc/s).

As expected, the layers progressively apply a filter to the signal. In particular, in the case represented in Figure 4.4, the resulting effect is that of a cascade of low-pass filters, with a decreasing trend of the spectral centroid as more layers are taken into account. More in general, the specific trend of this behavior might depend on the combined effect of the reservoir hyper parameters, IP adaptation and characteristics of the input signal. Indeed, on the real-world tasks presented in the next Chapter in Sections 5.3.1 and 5.3.2 we show also cases of high-pass filters. Moreover, as we can see in Figure 4.4 for the considered case, the effect of filtering clearly tends to decrease, with the shift of the spectral centroid approaching convergence as the depth of the recurrent architecture increases. Interestingly, in the case of FBC task convergence is achieved on the 7-th layer, i.e. in correspondence to the optimal number of layers found for the task (see Figure 4.2). Furthermore, from Figure 4.4, we note that, after convergence, some small fluctuations of the spectral centroid can be observed, which can be due to possible numerical errors of FFT operations and to the characteristics of the input signals (e.g., for the MuseData and Piano-midi.de tasks in Section 5.3.1 the convergence is smoother). In this respect, the adoption of $\eta = 0.01$ empirically shows to be a safe threshold value that is not too small to

follow the fluctuations neither too large to determine an insufficient depth (too far from the convergence). Apart from the specific trend of the spectral centroid for increasing number of layers, the same qualitative considerations made here generally apply also for the real-world tasks in Sections 5.3.1 and 5.3.2.

Overall, the properties of stacked filters are empirically evident in our experimental analysis that shows the convergence of the shift of the spectral centroid in both controlled scenario and real-world cases considered in this thesis, and the non-critical role of the η value for the generality of the algorithm. At the same time, our investigation shows an analysis methodology (i.e. to follow the trend of the spectral centroid on the plot over the layers, similarly to Figure 4.4) that can be helpful in assessing the useful (possibly different) η choice for the task at hand.

4.3 Discussion

The fundamental goal of the work presented in this Chapter consisted in the development of a design strategy for automatizing the choice of the number of layers in DeepESNs. Essentially, we exploited the idea that each new layer should provide an internal state representation that, in terms of frequency spectrum, is sufficiently diversified with respect to those developed in the previous ones. Collectively, this ensures that the dynamical component of the network provides an advantageous trade-off between the richness of temporal information representation (multiplicity of temporal scales) and the resulting complexity (final number of layers).

The outcomes of our experimental analysis showed that the proposed design strategy is effective in the RC context, leading to DeepESN setups that on the one hand are able to fruitfully exploit the depth in the comparison with shallowESN counterparts, and on the other hand outperform previous state-of-the-art results achieved by fully trained RNN on real-world problems (see Sections 5.3.1 and 5.3.2 contained in the next Chapter).

From the point of view of filtering, extending the analysis presented in Sections 3.3 and 3.4, the qualitative analysis on the considered tasks empirically showed that recurrent layers of a DeepESN architecture with IP adaptation can not only act as low-pass filter, but also as high-pass filter (see Sections 5.3.1 and 5.3.2). This allows the higher layers to develop faster state dynamics. We believe that these observations can stimulate further analytical/theoretical studies on the characterization of the filtering effect operated by a recurrent layer, in particular, focusing on the hyper parameters of reservoirs in hierarchical architectures. Interestingly, the developing

of faster state dynamics in higher layers of a stack of RNNs is observed also in the context of spiking neural networks [215].

The exploitation of our contribution can be also considered from the point of view of studies on the initialization and architectural properties of fully trained multi-layered neural architectures with back-propagation (stochastic gradient descent approaches). This is particularly interesting in consideration of the difficulties that are typically encountered in training deep networks [70], especially for studies regarding RNNs with ESN-based initialization, see [182]. Indeed, initialization approaches based on pre-training analysis can influence the efficiency and the stability of the convergence of gradient based algorithms in deep nonlinear networks [94, 162].

Investigations conducted in this work also allowed to address other research issues arising hot debates in the neural networks community. In particular, two relevant instances of such questions regard the performance comparison between deep and shallow RNN models [219, 72, 145, 73]. Through experimental comparisons between DeepESNs and ESNs, we practically demonstrated, at least in the considered tasks, the performance advantages that inherently stem from a suitable multi-layered organization of the recurrent part of the model (in the RC framework, i.e. taking aside the learning algorithms aspects of the recurrent part). At the same time, the possibility to effectively exploit the layering factor in the design of multi-layered recurrent networks, using the approach proposed in this thesis, paves the way for a grounded comparison between deep and shallow RNNs also in the context of trained models. Additionally, the results contributing to settle deep recurrent approaches, further encourage future analysis aimed at exploiting properly designed deep RNNs in modeling temporal information with latent compositionality under a generative setting.

4.4 Conclusions

In this Chapter, we have introduced a novel approach to address a fundamental issue in Deep Learning for sequence processing, i.e. the problem of how to choose the number of recurrent layers in a deep recurrent architecture. Remembering the scope of the approach, namely that the input signals are featured by multiple time scales and that the differences in the time scales are important for the learning task at hand, we aim at exploiting such differences to tailor the layered architecture to the task. In turns, the trained output part of the model can exploit the differentiation

provided through the layering for the performance aims on the learning task.

Indeed, framing our work in the deep RC area and making use of frequency analysis tools, we have defined an automatic design algorithm for DeepESNs, aimed at exploiting as much as possible the differentiation of the temporal information representation naturally developed by recurrent hierarchies. Under the assumed conditions, the provided approach enables to choose the proper number of recurrent layers avoiding to perform the training of the readout part for each possible number of considered layers. As such, compared to a standard selection process, the proposed method allows to obtain a reduction of the time cost of model selection that scales with the number of layers.

On the experimental side, in order to asses the effect of the diversification of the frequency components enriching the state representation through layering, we have analyzed the approach on a controlled scenario with a synthetic task characterized by signals with a predefined multiple time-scales dynamics. Quantitative and qualitative analysis on such task revealed that, in the considered experimental setting, the proposed design method is able to choose a proper number of layers reaching a better performance compared to alternative configurations with a different number of layers or with a shallow recurrent architecture.

The results achieved on the considered tasks showed that DeepESNs designed by our automatic algorithm consistently improve the performance of shallowESNs counterparts under the same experimental settings (and ranges for the hyper-parameter values).

Moreover, in the next Chapter (see Sections 5.3.1 and 5.3.2) we assess the design method on challenging real-world tasks in the areas of music and speech processing. Noteworthy, the performance achieved by DeepESN compares well with the state-of-the-art results previously obtained by fully trained RNN-based models on real-world tasks and RC approaches on the speech recognition task. This, in turn, suggests that music and speech processing represent instances of applicative domains with multiple time-scales information that can benefit from the DeepESN approach.

In conclusion, we believe that the design method proposed in this work can contribute to, and further stimulate, the development of approaches aimed to a principled automatic design of deep reservoir architectures in an information-based fashion, i.e. through quantitative and qualitative analysis of the dynamics emerging in the layers of stacked recurrent models.

Applications of DeepESNs

The content presented in this Chapter is based on works published in [62, 60, 6, 64, 65, 59] (see Section 1.5 for details).

In this Chapter we first present new application solutions in the field of Health Informatics based on RC and DeepESN models. Then, we analyze and evaluate the DeepESN model on prediction and classification real-world tasks characterized by high-dimensional time series. For each application, we compare the DeepESN model with the shallow counterpart. Finally, we provide a comparison between DeepESN model and fully trained RNN approaches on challenging prediction tasks.

In Section 5.1, we first propose a novel automatic system for human balance estimation based on RC paradigm and then we compare the ESN approach with the DeepESN model on the benchmark task. In Section 5.2, we propose a new method for the diagnosis of Parkinson's disease based on DeepESN. In Section 5.3, we evaluate the approaches introduced in Chapter 4 for the design of DeepESNs on prediction and classification tasks characterized by high-dimensional time series. Finally, in Section 5.4 we compare the DeepESN model with fully trained RNNs typically used in DL community on polyphonic music tasks.

5.1 Automatic Berg Balance Scale Estimation

Here, we introduce an automatic system for the balance estimation in the field of Health Informatics. In Section 5.1.1, we explain motivations and main aspects of the considered medical application. In Section 5.1.2, we describe the procedure performed to develop real-world datasets used in this context, then, we present an overview of the proposed system, and finally, we evaluate the RC model on the benchmark task. In Section 5.1.4, we compare the DeepESN model with the shallow counterpart on the benchmark task. Finally, Section 5.1.5 draws the conclusions.

5.1.1 Introduction

All European countries are experiencing aging of their populations, with a decrease in the number of people of working age per retiree. By 2050, an estimated 35% of the European population will be over the age of 60, compared to 20% in 2005. Health trends among older people are mixed: severe disability is declining in some countries but increasing in others, while mild disabilities and chronic diseases are generally increasing. The aging process is characterized by a constant decline of body functions and is frequently associated to a series of impairments involving reduction in mobility and cognitive decline [188]: these aspects work synergistically increasing the risk of falls. Prevention of falls should be one of the first defense lines to support an active aging. Accordingly, the balance assessment of elderly is assuming great relevance in clinical practice, with the development of several screening tools and tests that are used to assess stability or its deterioration: these include both simple clinical measures and also sophisticated technologies [25]. One of the common and easiest functional tests frequently used in medical practice is the Berg Balance Scale (BBS) test. Initially, this was proposed for balance assessment in elderly population but it has been frequently used in subjects with stroke [19], Parkinson's disease [152], brain injury [104], and multiple sclerosis [26]. The test is composed by 14 items, in the following also referred to as *exercises*, with a score ranging from 0 to 4 points. The maximum BBS score is 56 and the test duration time is $\approx$ 15-20 minutes. A score of 45 is indicated as a threshold for subjects at high risk of fall [153]; each reduction of 1 point in BBS score is correlated to an increased risk of 6-8% to fall [173].

Recently, within the aims of the DOREMI European project [5], a technological platform to support and motivate elderly people to perform physical activity has been developed, targeted at a reduction in sedentariness, cognitive decline and malnutrition, at the same time promoting an improvement in the quality of life and social inclusion. This is focused on the development of a systemic solution for healthy aging able to prolong the functional capacities of the elderly. One of the main innovative objectives of the DOREMI platform consists in the development of an automated system for balance assessment. In the proposed approach, the balance assessment system is an easy-to-use, cost-effective and unobtrusive ICT solution for early pre-frail risk detection and frailty prevention. This innovative DOREMI solution leverages the Wii Balance Board, a low-cost, portable and widely available force platform, able to evaluate the user weight distribution at the four corners of

its surface, developed by Nintendo for the Wii gaming console. This device has been compared to laboratory-grade force platforms [32, 114] and its accuracy proved to be acceptable for the employment in numerous scientific studies involving balance assessment [32, 214] and gait or balance rehabilitation [69, 172, 39]. The relation between statistical features and the evaluation of the balance stability in elderly patients has been investigated in [126], which presents a comparative study on stability-related measures using regression methods on data collected from a force platform with the aim of estimating the probability of the patient's fall. However, it is worth to note that the whole temporal signal generated by a force platform, such as the Wii Balance Board, potentially contains richer information than the above mentioned static features, thereby allowing to envisage approaches that try to directly and automatically exploit such richness of signal dynamics.

Here, we propose a novel system for automatic assessment of balance abilities in elderly, able to estimate the overall BBS score of a user based on the stream of input signals gathered from the Wii Balance Board during the execution of only one BBS exercise out of 14. The major scientific goal of this study is to assess the feasibility of accurately estimating the overall BBS score by exploiting the temporal series from pressure sensors gathered during a single exercise execution by the user, and to provide an experimental validation of the proposed system on real-world data.

Such a scientific challenge requires to address the fundamental questions of whether such temporal series contain enough information to be correlated with the full BBS score and whether a RNN model can efficiently exploit such an information to automatically estimate the score. For this purpose, we resort to RC models (described in Section 2.4.1), which are widely recognized as particularly appropriate for processing and extracting relevant dynamic knowledge from noisy temporal data. Finally, we compare the DeepESN model (introduced in Chapter 3) with the shallow RC counterpart (a 1-layered DeepESN with the same number of total recurrent units) on the balance assessment benchmark in order to evaluate the practical relevance in using layered RC architectures in this domain.

5.1.2 Balance Assessment Benchmark

Datasets

The datasets are collected during the experiments through an electronic balance called Wii Balance Board. It is a gaming device developed by Nintendo for the

Wii console. The device is a force platform with a width of 32,5cm, length of 52 cm and height of 7,5cm, weighing $\approx$ 4 kg, similar to a household body weight scale. The device is able to evaluate vertical forces at the four corner ends of its surface, i.e front left (FL), front right (FR), back left (BL), back right (BR), by using four strain-gauge pressure sensors placed inside the 4 feet bumpers. Even if the Wii Balance Board is considered a fairly cheap force platform, the device has been employed in numerous scientific studies involving balance assessment [32, 214] and rehabilitation addressing several disabilities [69, 172, 39], being a portable and widely available device with acceptable accuracy.

In the proposed system, the user BBS score is estimated from data generated by the execution on the Wii Balance Board of a single BBS exercise. For the design and validation of our system we restricted our attention on 3 out of the 14 exercises of the complete BBS test, namely exercises #6, #7, and #10. More details on the recruitment process and the protocols used during the measurement campaign are reported in [6].

For each time step t, the obtained signal values gathered by the balance board are collected into a column vector $\mathbf{u}(t) = [u_{\text{FL}}(t)\, u_{\text{FR}}(t)\, u_{\text{BL}}(t)\, u_{\text{BR}}(t)]^T \in \mathbb{R}^4$. Further details regarding pre-processing step are described in [6].

An example of the temporal signals gathered in correspondence of a complete exercise execution (including the phases of getting on and off the balance board) is reported in Figure 5.1, which illustrates the temporal evolution of the user weight values at the four corners of the balance board during the execution of exercise #10. Figure 5.1 shows how difficult would be to identify a pattern for the specific BSS score by human visual inspection, also due to the noisy nature of the signal. This is further complicated by the different ways the same exercise can be executed by the user, e.g. the way they step up onto the balance board (in terms of which foot is used first to step up and step down the board), the physical conditions of the users that can lead to lurching during the exercise, and the total duration of the execution.

As a result of the measurement campaign and data pre-processing we obtained Balance datasets for the definition of 3 regression tasks on sequences, one for each BBS exercise considered. The datasets contain couples of the type $(\mathbf{s}, y_{\text{target}})$, where $\mathbf{s} = [\mathbf{u}(1)\mathbf{u}(2)\ldots\mathbf{u}(L)]$ is the pre-processed 4-dimensional input sequence of length L containing the stream of weight values recorded by the balance board during the exercise execution, and $y_{\text{target}} \in [0, 56]$ is the corresponding target BBS score of the user, representing the ground-truth information evaluated by a clinician during the

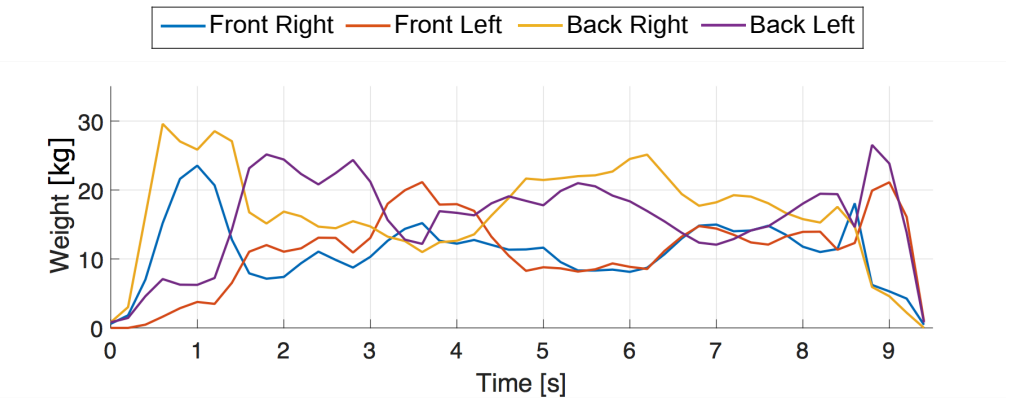


Figure 5.1: Plot of the pre-processed data from the BBS exercise #10. The exercise #10 is performed by the user placing the right foot first while stepping up onto the balance board and the right foot first while stepping down.

measurement campaign (by summing up the scores for all the 14 BBS exercises). Overall, the Balance datasets contain a total number of 470 sequences. The Balance datasets and their description are made publicly available at [10].

Learning BBS score using RC

The overall operation of the proposed system for automatic BBS score estimation is graphically sketched in Figure 5.2. While a subject executes a BBS exercise on the Wii Balance Board, the sensor stream is gathered and collected into a database as above described. Then the data is used as input for the neural network model that computes the overall BBS score estimate.

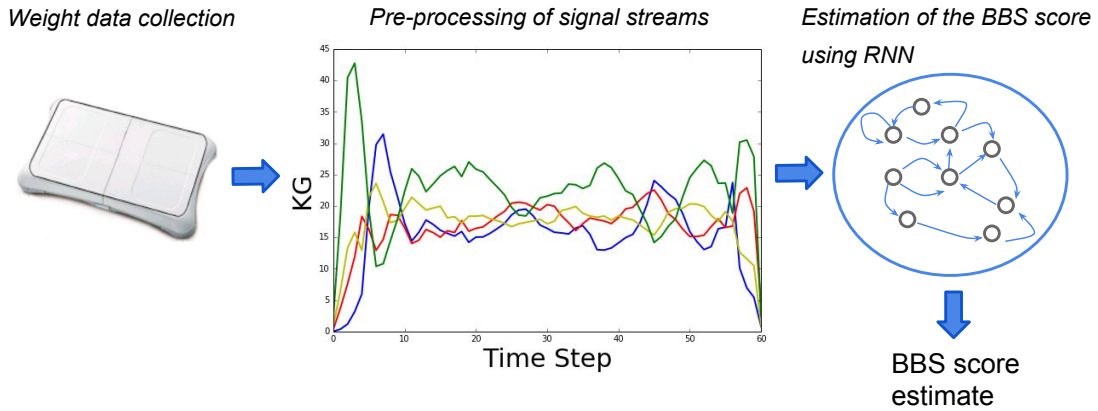


Figure 5.2: Graphical sketch of the overall operation of the proposed system for automatic BBS score estimation.

The balance signals are processed by RNNs, modeled according to the RC paradigm. Within the RC paradigm, we take into consideration the ESN, which is

an effective yet extremely efficient neural network model for learning in temporal domains. In particular, in this approach we use LI-ESN (described in Section 2.4.1).

For regression tasks in which only one output value is required in correspondence of an entire input sequence, such is the case of this application to the BBS score estimation, the output corresponding to $\mathbf{s}$ is computed from a state $\chi(\mathbf{s}) \in \mathbb{R}^{N_R}$ that encodes the entire input sequence as processed by the reservoir. Here, we consider the approach *root state mapping* (see Equation 2.9) in which the last reservoir state computed for $\mathbf{s}$ is considered as representative of the whole encoding process.

Experimental Setting

The predictive performance of the learning models taken into account is evaluated through a 7-fold cross validation process, splitting the available data according to a 3 persons-out approach, i.e. such that each fold contains all the data pertaining to 3 out of the 21 volunteer users. Note that the persons-out approach is of particular relevance for the purposes of this type of real-world applications as it allows us to estimate the performance of future assessments on new subjects during the real operational use of the proposed system (and thus completely unseen in the training phase).

In our computational experiments, we used LI-ESNs with 10% of reservoir connectivity and spectral radius $\rho = 0.99$. The values of the other relevant LI-ESN hyper parameters, including the reservoir dimension N_R, the input scaling $scale_{\text{in}}$, the leaking rate a, and the readout regularization for ridge regression training λ_r, were chosen from the ranges reported in Table 5.1 through a model selection process on a validation set, by means of an extra level of 6-fold cross validation on the training set of each external fold. For each reservoir hyper-parametrization, we independently generated 5 reservoir guesses, and the predictive performance in the different cases has been averaged over such guesses.

Hyper-parameter	Values considered for model selection
reservoir dimension N_R	10, 20, 50, 100, 200
input scaling $scale_{\text{in}}$	0.1, 0.5, 1
leaking rate a	0.1, 0.3, 0.5, 0.7, 1.0
readout regularization λ_r	0.001, 0.01, 0.1, 1, 10

Table 5.1: Range of LI-ESN hyper-parameters values considered for model selection.

We considered 3 regression tasks (see Section 5.1.2) on the data collected during

the execution of exercises #6, #7 and #10 of the BBS test, in the following referred to as *BBS-6*, *BBS-7* and *BBS-10*, respectively. The performance achieved by the learning models on the considered regression tasks has been computed by means of the Mean Absolute Error (MAE) and of the correlation coefficient R, respectively providing a measure of the absolute deviation and of the strength of linear relationship between the estimated and the ground-truth BBS score. Assuming the dataset under consideration contains N sequences, MAE and R are defined by the following equations:

$$MAE = \frac{1}{N} \sum_{\mathbf{s}} |y_{\text{target}}(\mathbf{s}) - y(\mathbf{s})|, \tag{5.1}$$

$$R = \sqrt{1 - \frac{\sum_{\mathbf{s}} (y_{\text{target}}(\mathbf{s}) - y(\mathbf{s}))^2}{\sum_{\mathbf{s}} (y_{\text{target}}(\mathbf{s}) - \frac{1}{N} \sum_s y(\mathbf{s}))^2}}, \tag{5.2}$$

where in correspondence of each input sequence $\mathbf{s}$, $\mathbf{y}(\mathbf{s})$ and $\mathbf{y}_{\text{target}}(\mathbf{s})$ denote the output of the learning model and the (ground-truth) target, respectively.

A preliminary experimental analysis of LI-ESN results on BBS-6, BBS-7 and BBS-10 allowed us to choose some common aspects of the experimental setting concerning the input sampling frequency and normalization. On the basis of preliminary results, we sampled the signals at a frequency of $5Hz$. Moreover, we normalized each signal dimension in each input sequence $\mathbf{s}$ to zero mean and unitary standard deviation (see [6] for further details).

Experimental Results

As described in Section 5.1.2, exercises #6, #7 and #10 of the BBS test were chosen by clinical staff for reasons of safety and feasibility of execution on the Wii Balance Board, while in the system for automatic balance assessment we require the user to perform only one BBS exercise. Thereby, with the aim of selecting the specific BBS exercise to be performed, we compared the predictive performance achieved by LI-ESN on the 3 tasks BBS-6, BBS-7 and BBS-10. Table 5.2 reports the values of MAE and R obtained on the BBS-6, BBS-7 and BBS-10 tasks by LI-ESN.

Task	TR MAE	VL MAE	TS MAE	TS R
BBS-10	3.56 ± 0.12	4.21 ± 0.14	4.80 ± 0.40	0.68
BBS-7	3.74 ± 0.11	4.74 ± 0.17	5.05 ± 0.32	0.51
BBS-6	4.43 ± 0.13	5.04 ± 0.19	5.53 ± 0.43	0.53

Table 5.2: Training (TR), validation (VL) and test (TS) MAE obtained by LI-ESN on the BBS-6, BBS-7 and BBS-10 tasks. R values on the test set are reported as well.

From such results we can see that the best performance on validation set was achieved on the BBS-10 task. Interestingly, the BBS-10 task also corresponded to the smallest generalization MAE and the larger R value. Overall, the results in Table 5.2 provides an experimental evidence that BBS exercise #10 enables a more accurate estimation of the total BBS score, thereby in order to develop our automatic system we restrict our focus on task BBS-10 only. Moreover, we consider the task BBS-10 as a benchmark to compare the LI-ESN approach with the DeepESN model in Section 5.1.4. Thereby, we evaluate the impact of layering on the model's performance in a basic balance assessment task regardless of the specific configurations for the development of the automatic system within the DOREMI project.

The applicative details regarding the development and the evaluation of the RC system for the clinical analysis of human balance assessment are presented in the next session.

5.1.3 RC System for BBS Score Estimation

In order to build up the automatic RC system for BBS score estimation, here, we select the final model performing a series of experimental evaluations on ad-hoc solutions developed to improve the clinical analysis of human balance assessment. In particular, we evaluate a weight sharing technique and the use of clinical data in addition to the time-series data gathered from the balance.

Weight Sharing Approach on Input Connections

During the exercise execution, the user can get on and off the balance board with the right or with the left foot in an arbitrary manner. The way in which the user gets on and off the board affects the shape of the input signals at the beginning and at the end of each input sequence (see e.g. Figure 5.1 for a graphical

example). This type of information is only related to the measurement campaign and the corresponding real-world data sampling, and it is not relevant to the aim of postural balance assessment. Accordingly, we do not want the learning models to specialize on this information. A neural network architectural variant that can be adopted to reach this purpose makes use of a weight sharing (WS) approach applied to the input-to-reservoir connections, such that the influence of the input signals coming from the left and right sides of the board is the same. Considering a row-wise aggregation of the input weights pertaining to the signals at the 4 corners of the balance board, the input-to-reservoir weight matrix can be written as $\mathbf{W}_{\text{in}} = [\mathbf{W}_{\text{FL}}\mathbf{W}_{\text{FR}}\mathbf{W}_{\text{BL}}\mathbf{W}_{\text{BR}}]$. In this case the adopted WS approach, graphically depicted in Figure 5.3, consists in sharing the weights in $\mathbf{W}_{\text{in}}$ such that $\mathbf{W}_{\text{FL}} = \mathbf{W}_{\text{FR}}$ and $\mathbf{W}_{\text{BL}} = \mathbf{W}_{\text{BR}}$. The performance achieved by LI-ESN on

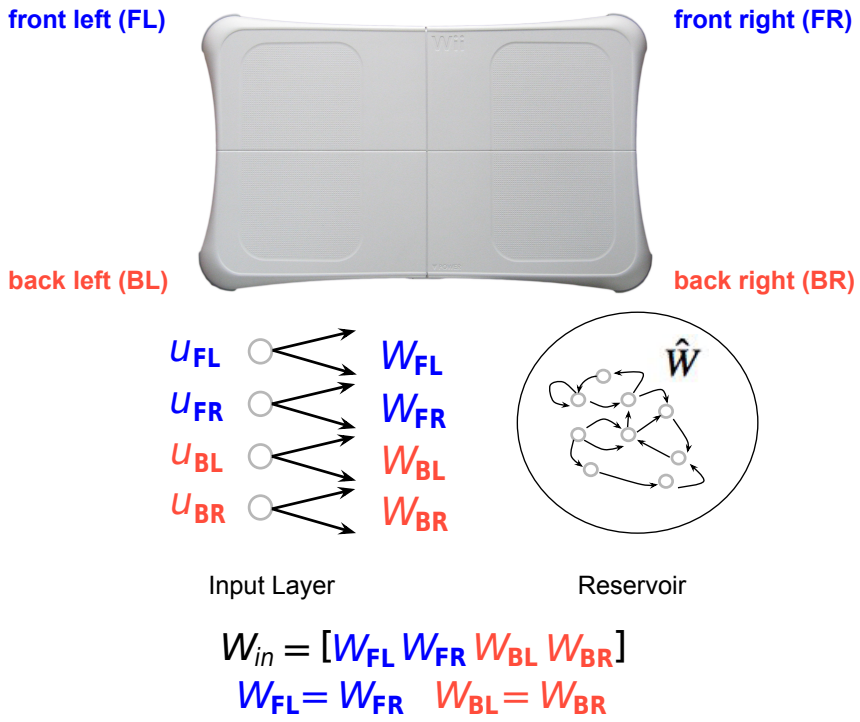


Figure 5.3: Graphical illustration of the adopted weight sharing approach. The input weights pertaining to signals coming from the left and the right side of the balance board are shared.

the BBS-10 task by adopting the WS technique is reported in Table 5.3. The positive effect of the WS approach in this application is testified by the fact that the predictive performance with respect to the case in which WS is not adopted is improved both on the validation and on the test sets. Indeed, through a comparison between Tables 5.3 and 5.2 it can be seen that the validation MAE is reduced by

0.12 BBS score points (0.21% over the whole BBS score range), corresponding to an average error reduction of 2.85% , while the test MAE is reduced by 0.77 BBS score points (i.e. 1.38% of the BBS score range), leading to an average error reduction of 16.04%.

Task	TR MAE	VL MAE	TS MAE	TS R
BBS-10 + WS	3.43 ± 0.04	4.09 ± 0.08	4.03 ± 0.13	0.71

Table 5.3: Training (TR), validation (VL) and test (TS) MAE obtained by LI-ESN on the BBS-10 task using the WS technique. The R value on the test set is reported as well.

Use of Clinical Data

A further significant experimental assessment consisted in the evaluation of the influence on the predictive performance of users' clinical data such as height, weight, age and gender. The values of such clinical data were used as input to the model by considering the height expressed in meters, the weight expressed in hundreds of kg (i.e. dividing the weight in kg by 100), the age expressed in hundreds of years (i.e. dividing the age in years by 100), and the gender expressed as a binary value (-1 for women and $+1$ for men)[1].

To avoid the introduction of artifacts related to the specific dataset on which we trained the learning models, we excluded the use of the clinical features that resulted in a high correlation with the BBS scores of the users in our sample data (without however having a known correlation in the general case or in literature). This analysis allowed us to exclude from consideration users' height and gender, restricting the focus on users' weight and age only. The augmented input has been implemented by appending at each time step of each sequence in the dataset the value of the clinical parameter of the corresponding user as a further input element. This process resulted in input sequences consisting of 5 elements per time steps, i.e. $\mathbf{u}(t)_{\text{augmented}} = [u_{\text{FL}}(t)u_{\text{FR}}(t)u_{\text{BL}}(t)u_{\text{BR}}(t)u_{\text{clinical}}]^T \in \mathbb{R}^5$, where u_{clinical} is the clinical feature (weight or age). We accordingly prepared two variants of the BBS-10 task, corresponding to the cases of augmenting with users' weight or age, in the following referred to as tasks BBB-10-W and BBB-10-A, respectively. Table 5.4 reports the performance achieved by LI-ESN on these two tasks, showing that the best result is achieved in correspondence of the BBS-10-W task. Comparing Tables 5.4

[1]These choices were made as simple scaling approaches, in order to have values approximately in the same range of the signals coming from the balance board.

and 5.2, it is possible to observe that augmenting the input data with the user weight information ultimately leads to a reduction in the validation MAE of 0.13 BBS score points (0.23% of the BBS score range), corresponding to an average error reduction of 3.09%, whereas the improvement in the test MAE is quantifiable in 0.18 BBS score points (the 0.32% of the BBS score range), with an average error reduction of 3.75%.

Task	TR MAE	VL MAE	TS MAE	TS R
BBS-10-W	3.50 ± 0.08	4.08 ± 0.09	4.62 ± 0.30	0.69
BBS-10-A	3.67 ± 0.12	4.23 ± 0.13	4.52 ± 0.27	0.65

Table 5.4: Training (TR), validation (VL), test (TS) MAEs, obtained by LI-ESN on the BBS-10-W and BBS-10-A tasks. R values on the test set are reported as well.

Joint Use of Weight Sharing and Clinical Information

The results discussed above, have shown the practical advantage in terms of improved performance on the clinical analysis of human balance assessment brought about by the use of an appropriate WS technique, or when the learning model receives in input the overall user weight in addition to the time series data gathered by the balance board. In this Section, we explore the synergy of these two approaches to provide a final LI-ESN setup that has to be implemented in the RC system for BBS score estimation. Accordingly, Table 5.5 shows the results obtained by LI-ESN using the WS approach on the BBS-10-W task, i.e. augmenting the input with the user weight information. Table 5.5 also recalls the performance achieved by LI-ESN without the WS approach and the augmented input information to facilitate performance comparison.

Task	TR MAE	VL MAE	TS MAE	TS R
BBS-10 (without WS)	3.56 ±0.12	4.21 ± 0.14	4.80 ± 0.40	0.68
BBS-10-W (with WS)	3.11 ± 0.05	3.85 ± 0.08	3.80 ± 0.17	0.76

Table 5.5: Training (TR), validation (VL) and test (TS) MAE obtained by LI-ESN on the BBS-10 task (without WS) and on the BBS-10-W task (with WS). R values on the test set are reported as well.

Results in Table 5.5 show that the joint effect of using WS and user weight input information is indeed superior to the single improvements due to the use of the two approaches alone. Indeed, LI-ESN with WS and users' weight in input achieved very close values of validation and a test MAE, respectively equal to 3.85 ± 0.08

and 3.80 ± 0.17, corresponding to an improvement of 0.37 BBS score points in the validation set (0.66% of the BBS score range), i.e. an average performance improvement of 8.79%, and of 1 point in test set (1.79% of BBS score range), with an average performance improvement of 20.83%. Moreover, the R value on the test set is overall improved of 0.08, i.e. of the 11.76%.

Selected Model and Analysis of BBS score Estimation on Subjects

The experimental evaluations above presented in this section, allowed us to select the final setting of the RC model that has to be implemented in the automatic system for the clinical analysis of balance assessment. In particular, for this purpose we selected LI-ESN with WS using board sensor data from exercise #10 and complemented by the subject personal weight information. The predictive performance results achieved in the final setting are summarized in Table 5.6.

Model	VL MAE	TS MAE	STDg	STDs	STDf	STDu	TS R
LI-ESN	3.85	3.80	0.17	2.92	1.64	2.01	0.76

Table 5.6: Performance results achieved in the final setting for BBS score estimation, i.e. with LI-ESN using WS on the BBS-10-W task. The table reports validation (VL) and test (TS) MAE, along with the standard deviation computed on the test set with respect to: the reservoir guesses (STDg), the different sequences (STDs), the external folds of the double cross-validation scheme (STDf), the different users (STDu). The R value on the test set is reported as well.

As can be seen, the selected model achieved a validation MAE (mean of errors over the folds of the cross-validation) of 3.85 BBS score points (corresponding to the 6.88% of the total BBS score range), a test MAE of 3.80 BBS score points (corresponding to the 6.79% of total BBS score range). Such a result is indeed extremely good, considering that the generalization error is even below the threshold of 4 BBS score points, that is the score range of a single BBS exercise. Moreover, as seen in Table 5.6, the selected LI-ESN achieved a standard deviation of the MAE of 0.17, 2.92, 1.64 and 2.01 BBS score points with respect to the reservoir guesses, the different sequences (i.e. different exercise repetitions by the same user), the external folds in the double cross-fold validation and the different users, respectively. Note that these results are largely within the range of tolerance for clinical interpretation. A recent study [34] has, in fact, estimated that it is necessary to observe a difference of a least 8 BBS score points in order to diagnose an actual change in the postural balance ability of a subject.

The quality of the automated BSS estimate can be appreciated at a subject-level by graphically summarizing the results for all users in test. In this sense, Figure 5.4 shows a plot comparing the ground-truth BBS score measured by the clinicians for each subject versus the corresponding estimate provided by the LI-ESN model (computed by averaging the results obtained for each exercise repetition by the subject and for each reservoir guess). As can be seen, the points in the plot are generally distributed close to the bisector of the x-y axis, with MAE of 3.36 BBS score points, a correlation coefficient R equal to 0.8257, with p-value $p < 0.0001$. These summarized results, looking also at the distribution of errors, confirm the good quality of our RC-based system for automatic BBS estimation.

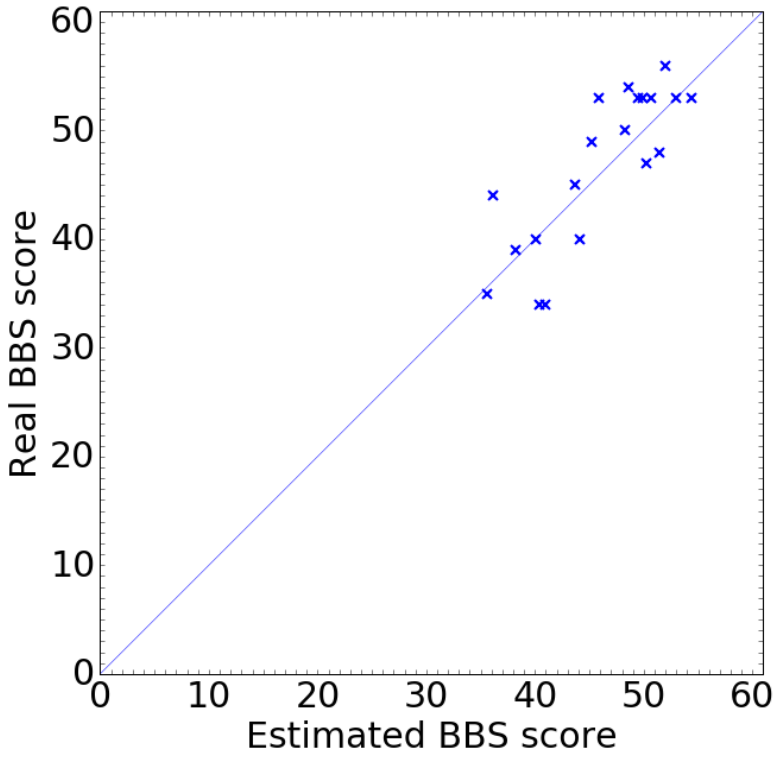


Figure 5.4: Real (ground-truth) versus LI-ESN estimated BBS scores plot (each point corresponds to a different user, evaluated in the test set).

Comparison with the state of the art

We compared RC setting selected for the proposed automatic system with other basic neural network approaches (see the experimental comparison described in [6]). Interestingly, the RC paradigm resulted the best applicative solution for the estimation system in terms of accuracy.

Overall, it is worth mentioning that the proposed system compares well with the

ones already reported in literature for BBS score estimation tasks. In particular, the work in [176] describes a system based on a k-NN algorithm using data gathered during the execution of 3 BBS exercises from a 3-axial accelerometer positioned on the lower back of the user, achieving a MAE of 4.63 ± 3.89. The system described in [8] uses data gathered from 5 body-fixed sensors during the execution by the user of the entire BBS test and proposes to estimate the overall BBS score by summing up individual estimations of the 14 BBS exercises scores obtained by MLPs, obtaining a MAE of $\approx 1.5 \pm 1$. In comparison to such approaches our system outperforms the predictive performance of the work in [176], while the one in [8] results to be superior. In both cases, however, note that the literature systems present a much higher intrusiveness, requiring the user to wear physical devices (1 sensor in [176], 5 sensors in [8]) and to perform a higher number of balance exercises (3 BBS exercises in [176], all the 14 BBS exercises in [8]). Thereby, the system proposed in this work, using data collected during the execution of a single BBS exercise from an unworn device (a balance platform) is characterized by a favorable trade-off between predictive performance and intrusiveness.

5.1.4 Comparison with DeepESN model

In this section, we compare the DeepESN model with the LI-ESN approach, that here we call shallowESN (i.e., a 1-layered DeepESN with the same number of total units), on the benchmark task BBS-10 described in Section 5.1.2. The use of a benchmark task allows us to evaluate the practical relevance of using layered recurrent architectures in this domain regardless of the specific application settings adopted for clinical motivations in the implementation of the estimation system proposed in Section 5.1.3.

As in the setting of the benchmark task described in Section 5.1.2, here, the output of the considered networks is computed through root state mapping function (see Equation 2.9). For the DeepESN architecture we considered a number of recurrent layers $N_L = 10$. The models are individually optimized with a grid search approach on hyper-parameters values as specified in Table 5.7. For each hyper-parametrization, we generated 5 reservoir guesses, averaging the results on such guesses.

Hyper-parameter	
total recurrent units	$50, 100, 150, 200, 250$
readout regularization λ_r	$0.001, 0.01, 0.1, 1, 10$
input and inter-layer input scaling σ	0.1, 0.5, 1, 1.5, 2
leaking rate a	0.1, 0.3, 0.5, 0.7, 0.9, 1.0
spectral radius ρ	0.1, 0.3, 0.5, 0.7, 0.9, 1.0

Table 5.7: Range of DeepESN and shallowESN hyper-parameters values for model selection in BBS-10 task.

Table 5.8 shows the performance obtained by DeepESN and shallowESN on the BBS-10 task.

Model	TR MAE	VL MAE	TS MAE	TS R
DeepESN	4.13 ± 0.03	4.14 ± 0.04	4.48 ± 0.06	0.70
shallowESN	4.17 ± 0.02	4.18 ± 0.03	4.52 ± 0.06	0.69

Table 5.8: Training (TR), validation (VL), test (TS) MAEs, achieved by shallowESN and DeepESN on the BBS-10 task. R values on the test set are reported as well.

The results show that DeepESN outperforms shallowESN by 0.04 MAE. The improvement of the performance achieved by DeepESN on this regression task resulted significative with a p-value of 0.0156 which is computed performing the Wilcoxon signed-rank test between DeepESN and shallowESN.

Although there are not big performance differences between DeepESN and ESN, the comparison performed on benchmark task suggests that layering in DeepESN model is able to improve the performance. Therefore, these results suggest that the application of DeepESN deserves to be taken into consideration for further researches in this field.

5.1.5 Conclusions

In this section we proposed an innovative learning system for the automatic assessment of balance abilities in the elderly. The main scientific challenge tackled by the study concerned assessing the feasibility of inferring the overall BBS score (based on the clinical evaluation on all the 14 BBS exercises) on the basis of the data streams collected by a Wii Balance Board during the execution by the subject of a single BBS exercise. This study represents a positive answer to such a question, showing that recurrent neural networks, modeled according to the principles of the

RC paradigm, allow to effectively yet efficiently exploit (in an automatic fashion) the richness of temporal dynamics contained in the data streams gathered from the balance board to extract information that is relevant for the task of overall BBS score estimation.

In order to train and experimentally validate the learning system, a measurement campaign has been carried out on 21 volunteer users, gathering data pertaining to the execution of a pool of 3 BBS exercises (i.e. #6, #7 and #10), considered as the most clinically suitable for the aims of automatic BBS score estimation from balance board data, and among these exercise #10 has been selected as the benchmark task. The resulting Balance dataset has been made publicly available and represents another relevant outcome of the work.

In order to develop an accurate system for the clinical analysis of balance assessment, the final setup of the system is obtained evaluating the model on the benchmark task considering the use of clinical data and weight sharing techniques. The experimental analysis of the proposed system based on the RC approach on real-world data, conducted by means of rigorous persons-out cross-fold validation, showed a good predictive performance, allowing to accurately estimate the overall BBS score with an average generalization error of 3.80 BBS points in test. Interestingly, such a value is below the 7% of the whole BBS range (0-56) and it is even smaller than the extent of the range of assignable points for each single BBS exercise (0-4), ultimately suggesting that, with regard to the execution of one BBS exercise, the learning models have been able to extract more information than the one that can be provided by a clinician in terms of the score of a single exercise.

The proposed system The relevance of the results achieved with RC networks has been further assessed through a performance comparison with other basic neural networks models, generally showing a better generalization performance with lower variability, and a favorable ratio among training, validation and test errors. Moreover, in comparison with literature works, our approach showed a favorable trade-off between predictive performance and intrusiveness.

Overall, the system proposed in this work compares well with the state of the art. The RC approach show better generalization performance with lower variability with respect to other literature approaches. Moreover, in comparison with literature works, our approach showed a favorable trade-off between predictive performance and intrusiveness. The system proposed in this work represents an automatic tool for the accurate estimation of a users' score at the BBS test (of $\approx$ 15-20 minutes of duration) from the execution of only one exercise of the test (of $\approx$ 10 seconds

of duration, i.e. $\approx 1\%$ of the duration of the whole test), thereby resulting in a tremendous time saving in the task of monitoring balance stability in elderly people. The system is characterized by limited obtrusiveness since it does not require the subject to wear any sensor. In this respect, it appears of straightforward use and particularly suitable for autonomous usage. Clinical and experimental evidence has in fact highlighted how the BBS exercise selected for our setup is of simple and safe execution and can be performed even without supervision by a clinician.

Finally, we performed further experimental evaluations on the benchmark task in order to compare the DeepESN model with the shallow counterpart. The results showed that the DeepESN model outperforms the shallowESN model in terms of performance. Therefore, DeepESN deserves to be considered as a good applicative solution for further works in this field.

5.2 Diagnosis of Parkinson's Disease

In this section, we present a novel application approach for the diagnosis of Parkinson's Disease based on DeepESN model. In Section 5.2.1, we briefly describe the application background. Moreover, in Section 5.2.2 we present the results achieved by our approach. Finally, in Section 5.2.3 we draw the conclusions.

5.2.1 Introduction

Parkinson's disease (PD) is a neurodegenerative disease that mainly affects the extrapiramidal motor system causing tremor, bradykinesia, rigidity and loss of postural reflexes [102]. The analysis of motor capacities such as handwriting and sketching abilities of patients is performed to assess and diagnose PD [161]. While handwriting abilities are influenced by language capacities, sketching abilities involved in the execution of a spiral test (i.e., drawing a spiral with a pen) are considered as measures independent of education [161]. An example of application of Deep Learning models for PD classification, applied directly to images of spiral tests, is introduced in [147]. In [217] it is introduced a study on PD diagnosis through statistical methods based on the analysis of velocity and the pen pressure data collected from a tablet device during the execution of the spiral test.

Here, we propose a novel approach for diagnosis of PD based on RNNs that are able to robustly exploit the whole time course of noisy and heterogeneous time-series data collected from a tablet device during the execution of spiral tests. Moreover,

we consider the Reservoir Computing (RC) framework (see Section 2.4.1) that obtained state-of-the-art results in the clinical assessment application presented in the previous Section 5.1 which is related to the study of neurological diseases. In Particular, in this application we aim to exploit the ability of DeepESN model in the processing of signal characterized by multiple time-scale dynamics (see Section 3.2) to for the study of PD through the analysis of tablet signals.

We assess our approach for diagnosis of PD on a publicly available dataset of spiral tests introduced in [93]. Moreover, in order to investigate the practical relevance of using layered recurrent architectures in this domain, the experimental analysis is also conducted in comparison to the shallowESN model. Finally, we evaluate the results achieved by ensembling the realizations of the selected model, in order to improve our application approach in terms of classification performance.

5.2.2 DeepESNs for Diagnosis of PD through Spiral Tests

We evaluated the proposed DeepESN model on the spiral dataset described in [93]. This dataset is publicly available on the UC Irvine Machine Learning Repository[2]. The dataset is composed by spiral tests executed on a tablet device by 61 PD patients and 15 control patients without PD. For each time-step, the time series gathered from the tablet contain pen position (x and y components), pen pressure and grip angle. In our method, the models were fed directly with such signals without feature extraction or data preprocessing. Note that, the whole temporal signal generated by the tablet potentially contains a richer information than static preprocessed features, thereby a RNN model can be more effective on the analysis of this kind of data. Figure 5.5 a), 5.5 b), 5.5 c) and 5.5 d) show (x,y) pen position, pen pressure and grip angle gathered at each time-step from tablet during the execution of the test by a control (Figure 5.5 a), 5.5 c)) and a PD (Figure 5.5 b), 5.5 d)) patient. As we can note, by visual inspection the sketches showed in Figure 5.5 a) and 5.5 b) are quite similar and consequently the classification task is not trivial in such cases. Moreover, from Figure 5.5 c) and 5.5 d) it is possible to see that the signals relative to pen pressure and grip angle are rather noisy, therefore, the analysis of such kind of data without feature extraction can be challenging since we need a noise-robust approach to perform a correct classification.

For the output computation of the network we considered *mean state mapping* (see Equation 2.10). We evaluated the generalized performance of the proposed

[2] <https://archive.ics.uci.edu/ml/datasets/Parkinson+Disease+Spiral+Drawings+Using+Digitized+Graphics+Tablet>

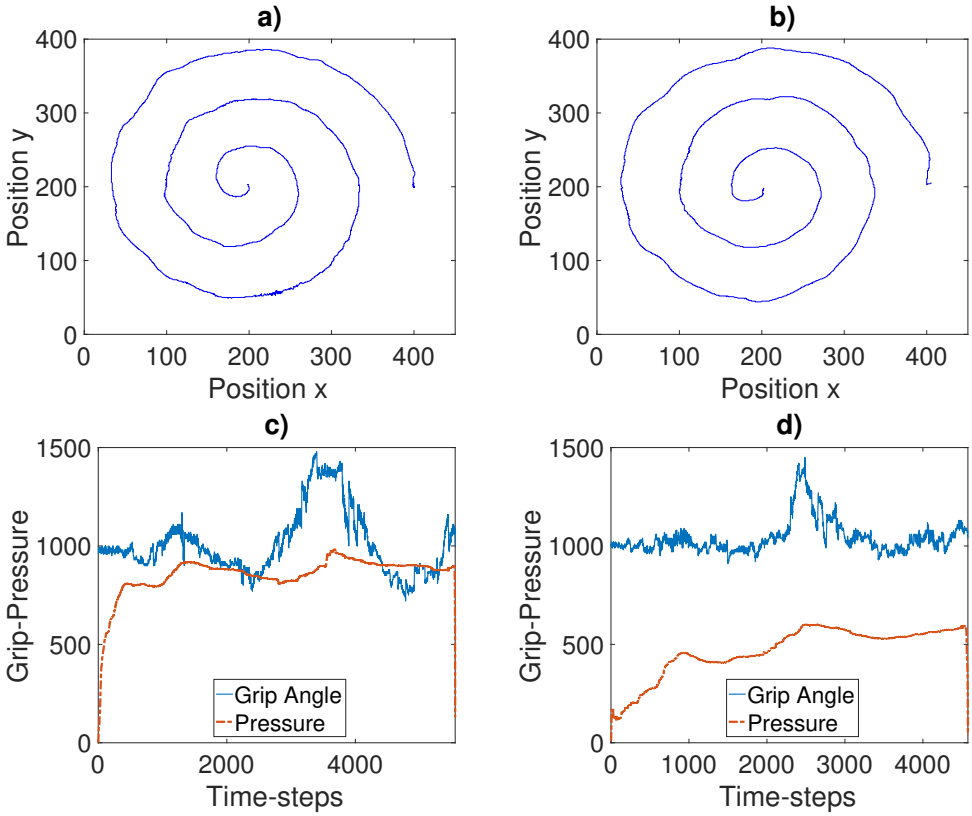


Figure 5.5: Pen positions (1st row), pressure and grip angle (2nd row) gathered for a control (Figure a), c)) and a PD (Figure b), d)) patient.

approach through a 3-fold cross validation, inserting in each fold 20 (or 21) PD and 5 control patients. We considered a 10-layered DeepESN (i.e. $N_L = 10$) with the same number of N_R units per layer and a leaky integrator of $a^{(l)} = 0.1$ for each layer l. The rest of the DeepESN hyper-parametrization were selected from the ranges reported in Table 5.9, on a validation set by an extra level of 5-fold validation on each fold. For each hyper-parametrization, we generated 10 reservoir guesses, averaging the results on such guesses.

Hyper-parameter	
recurrent units per layer N_R	$10, 20, 30, 40, 50$
input scaling σ	0.1, 0.5, 1, 2
inter-layer scaling $\hat{\sigma}$	0.1, 0.5, 1, 2
spectral radius ρ	0.7, 0.8, 0.9, 1.0
readout regularization λ_r	$0, 10^{-10}, 10^{-9}, ..., 10^{0}$

Table 5.9: Range of DeepESN hyper-parameters values for model selection.

Table 5.10 shows the accuracies and the standard deviations on reservoir guesses (in parenthesis) obtained by DeepESN and shallowESN in PD classification task.

In order to perform a fair comparison, we selected the shallowESN (i.e., 1-layered DeepESN) considering the same hyper-parameters ranges as for DeepESN and the same range of total number of recurrent units (i.e. 100-500), where the specific choice depends on the fold of the 3-fold cross validation. Interestingly, DeepESN

Model	TR	VL	TS
DeepESN	**94.27% (1.18%)**	**86.57% (2.62%)**	**87.20% (2.78%)**
shallowESN	91.60% (0.56%)	83.62% (3.39%)	84.13% (2.63%)

Table 5.10: Accuracies and standard deviations on reservoir guesses (in parenthesis) obtained by DeepESN and shallowESN in validation, training and test.

model outperformed shallowESN in training, validation and test set of 2.67%, 2.95% and 3.07% of accuracy respectively. This comparison result suggests that the ability of DeepESN in providing a hierarchical temporal representation of input signals with complex dynamics allows us to improve the performance on this kind of tasks. For completeness, we statistically compared the accuracies obtained on test set by DeepESN and shallowESN models through McNemar's test (a χ^2 variant). Classification accuracies obtained by DeepESN and shallowESN resulted significantly different with a p-value of 0.0032.

To exploit the variability provided by different reservoir guesses, we evaluated the selected model considering an ensemble approach. Accordingly, the classification was performed averaging the output of the different guesses of the selected model. Results are reported in Table 5.11, and show that the ensemble approach allowed DeepESN to achieve improved performance in training, validation and test set of 0.40%, 2.95% and 2.13% of accuracy respectively. Overall, our proposed automatic system obtained a test accuracy of 89.33% with a sensitivity (percentage of PDs correctly classified) and a specificity (percentage of controls correctly classified) in test set of 90.00% and 80.00% respectively.

Model	TR ACC	VL ACC	TS ACC	TS SEN	TS SPEC
DeepESN	94.67%	89.52%	**89.33%**	**90.00%**	**80.00%**

Table 5.11: Accuracy (ACC), sensitivity (SEN) and specificity (SPEC) obtained by ensemble of DeepESN in training (TR), validation (VL) and test (TS) set.

For what concerns the comparison with the state of the art, results of DeepESN in this experiments compare well with literature approaches recently devised on the same type of input data. In particular, in [217], a method is introduced for diagnosis of PD based on the analysis of spiral tests gathered from a similar tablet device, achieving a classification accuracy in PD identification of 79.1%.

Interestingly, our approaches outperform such method obtaining a test accuracy of 84.13%, 87.20% and 89.33% achieved by shallowESN, DeepESN and ensemble of DeepESNs respectively. This comparison further indicates that, contrary to what might appear at first glance from the example in Figure 5.5, simple statistics on the input signals, e.g. pressure data, are not rich enough to accurately discriminate PD[3], while our approach can capture relevant information from the whole temporal signal, allowing to effectively improve such results.

5.2.3 Conclusions

In conclusion, we proposed a novel approach for diagnosis of PD based on DeepESN model. The deep recurrent model is fed by the whole time series gathered from a tablet during the sketching of spiral tests. We performed a comparative assessment of our approach on a public dataset containing spiral tests executed by PD and control patients. Results showed that the predictive accuracy obtained by DeepESN significantly outperforms the result of shallowESNs, highlighting the potential benefits of deep recurrent architectures in the treatment of temporal signals for PD diagnosis. Moreover, the use of ensemble method on the selected DeepESN model led to a further performance enhancement. Overall, the proposed approach compared well also with respect to state-of-the-art results, further stressing the potentiality of exploiting the whole richness of temporal signals for PD diagnosis.

At the best of our knowledge, this work represents the first attempt to develop an approach for diagnosis of PD by using recurrent models, such as DeepESN, able to develop hierarchical temporal representations from tablet signals without the need of feature extraction and data preprocessing.

5.3 High-dimensional Time series: Prediction and Classification

In the previous sections of this Chapter, we focused our work on the introduction of original application solutions in the field of Health Informatics. This also allowed us to evaluate the practical relevance in using deep RC architectures in medical applications.

Here, we focus the studies on the analysis and the evaluation of the modeling approaches proposed in this thesis on real-world benchmarks characterized by

[3]Correlation between averaged pressure and PD in the dataset used in our work is weak (-0.31).

high-dimensional time series. Thereby, we experimental assess the DeepESN model introduced in Chapter 3 and the design approaches defined in Chapter 4 on challenging time-series prediction and classification real-world tasks.

In Sections 5.3.1 and 5.3.2 we present the experimental results obtained on polyphonic music and speech recognition tasks respectively comparing our approach with the applicative state of the art. Moreover, in Section 5.3.3 we experimentally analyze the filtering effect over the depth of deep RNNs using spectral analysis approaches (defined in Section 4.2) on polyphonic music and speech recognition tasks. Finally, in Section 5.4 we perform a model comparison between DeepESN and typical fully trained RNNs on 4 polyphonic music tasks which represent challenging real-world benchmarks for time-series prediction on high-dimensional sequences.

5.3.1 Polyphonic Music Tasks

Here, we evaluate our model on polyphonic music tasks defined in [23]. In particular, we consider two datasets, namely Piano-midi.de[4] and MuseData[5]. These datasets are characterized by high dimensionality and complex temporal dependencies involved at different time scales, forming interesting benchmarks for RNNs [13]. The two datasets are characterized by complex piano and orchestral compositions with a number of simultaneous notes that ranges from 0 to 15. The musical compositions are represented by piano-rolls that were preprocessed from MIDI files. Training, validation and test sets of preprocessed piano-rolls are available on the website[6] of the authors of [23]. Table 5.12 shows the main characteristics of Piano-midi.de and MuseData preprocessed piano-rolls as provided in [23].

Dataset	Split	# Samples	Avg len	Min len	Max len
Piano-midi.de	Training	87	872.5	111	3857
	Validation	12	711.7	209	1637
	Test	25	761.4	65	2645
MuseData	Training	524	467.9	9	3457
	Validation	135	613.0	63	3723
	Test	124	518.9	45	4273

Table 5.12: The main characteristics of the preprocessed piano-rolls samples in training, validation and test set, as defined in [23].

[4] Classical piano MIDI archive www.piano-midi.de
[5] Library of orchestral and piano classical music from CCARH www.musedata.org
[6] www-etud.iro.umontreal.ca/~boulanni/icml2012

In this representation, a musical composition is a sequence of 88- and 82-dimensional vectors for Piano-midi.de and MuseData tasks, respectively. In both cases, at each time-step a variable is set to 1 if the note is played and to 0 otherwise.

A polyphonic music task is a next-step prediction task on high-dimensional vectors. In particular, the aim of the tasks is to predict the notes played at time-steps $t+1$ (i.e., the vector $\mathbf{u}(t+1)$) given the notes played at time-step t (i.e., the vector $\mathbf{u}(t)$). In order to compare and evaluate the classification performance of the models, we measured the expected frame-level accuracy (FL-ACC) defined as in [11] and adopted in polyphonic music tasks in [23], computed as follows:

$$\text{FL-ACC} = \frac{\sum_{t=1}^{T} TP(t)}{\sum_{t=1}^{T} TP(t) + \sum_{t=1}^{T} FP(t) + \sum_{t=1}^{T} FN(t)}, \tag{5.3}$$

where T is the total number of time-steps of all sequences (i.e., musical compositions) considered for the evaluation and $TP(t)$, $FP(t)$ and $FN(t)$ are respectively the number of true positive notes, false positive notes and false negative notes predicted at time-step t (i.e., the vector $\mathbf{y}(t)$).

In our experiments on these tasks, we used reservoirs initialized with 10% of connectivity. For what regards DeepESN, the model selection process is performed by considering recurrent layers with a number of units per layer N_R varying in $\{50, 100, 200\}$. As regards readout training, we used ridge-regression with a regularization parameter λ_r in $\{0, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}\}$. We performed the design Algorithm 4 considering a number of maximum recurrent layers $M_L = 50$. All other aspects of experimental setup were as described in Section 4.2.3, and the remaining hyper parameters were chosen (for model selection purposes) from the same ranges as shown in Table 4.1. To assess the effectiveness of the proposed methodology, we compared the performance achieved by DeepESNs built using the method proposed in Section 4.2, with the one obtained by shallowESNs, considering the same ranges for hyper-parameters values shown in Table 4.1 and a range of total recurrent units in $1000 - 7000$ and $1000 - 7200$ (with a step of 200), for Piano-midi.de and MuseData respectively. Moreover, we compared the performance obtained by our model with the state-of-the-art approach that achieved the best FL-ACC results on the considered tasks [23], namely RNN-RBM. RNN-RBM is a sequence of Restricted Boltzmann machines (RBMs) whose parameters are the output of a deterministic RNN with proper constraint on the distribution of hidden units [23]. Other examples of applications that assess the performance of RNNs models using the FL-ACC measure on the considered polyphonic music tasks are

presented in [143], in which however the pre-processing of piano-rolls is different[7] and, as this affects the performance, the corresponding results are thereby difficultly comparable.

Model	Piano-midi.de	MuseData
DeepESN	**33.22 (0.12) %**	**36.43 (0.05) %**
shallowESN	31.76 (0.08) %	35.31 (0.03) %
RNN-RBM [23]	28.92 %	34.02 %

Table 5.13: FL-ACC (and standard deviation on reservoir guesses, shown in parentheses) obtained on the test set of the Piano-midi.de and MuseData tasks by DeepESN, shallowESN and RNN-RBM.

Table 5.13 shows the FL-ACC obtained on the test set by DeepESN, shallowESN and RNN-RBM on Piano-midi.de and MuseData tasks. In Piano-midi.de task, the hyper parameters that obtained the best performance on the validation set are $N_L = 35$, $N_R = 200$, $\rho = 0.1$, $scale_{\text{in}} = 0.1$, $a = 0.7$, $\sigma_{\text{IP}} = 0.1$ and $\lambda_r = 10^{-1}$ for DeepESN, and $N_L = 1$, $N_R = 5000$, $\rho = 0.5$, $scale_{\text{in}} = 0.01$, $a = 0.1$, $\sigma_{\text{IP}} = 0.1$ and $\lambda_r = 10^{-2}$ for shallowESN. While for what regards MuseData task, the hyper parameters that obtained the best performance on the validation set are $N_L = 36$, $N_R = 200$, $\rho = 0.1$, $scale_{\text{in}} = 0.1$, $a = 0.7$, $\sigma_{\text{IP}} = 0.1$ and $\lambda_r = 10^{-2}$ for DeepESN, and $N_L = 1$, $N_R = 6000$, $\rho = 0.3$, $scale_{\text{in}} = 0.01$, $a = 0.3$, $\sigma_{\text{IP}} = 0.1$ and $\lambda_r = 10^{-2}$ for shallowESN. Noteworthy, our approach achieved the best results on both the tasks, outperforming the state-of-the-art RNN-RBM model and leading to an improvement of the test accuracy of 4.30% and 2.41% for Piano-midi.de and MuseData, respectively. Moreover, DeepESN outperforms shallowESN with an improvement of 1.46% and 1.12% FL-ACC on Piano-midi.de and MuseData tasks, respectively.

These results highlight the effectiveness of the proposed design approach to manage complex high-dimensional time series on real-world tasks without re-training the readout for each configuration of the number of layers considered, outperforming at the same time the shallowESN.

We next analyzed how the choice of the number of layers performed by the design algorithm allows the DeepESN architecture to reach a good performance. To this end, we evaluated the quality of our design method by comparing the

[7] Table 5.12 shows the characteristics of preprocessed piano-rolls provided by the authors of paper [23] in which are defined the polyphonic music tasks. Note that the characteristics of preprocessed piano-rolls in Table 1 of paper [143] are different.

results obtained considering progressively more layers in the DeepESN architecture with the performance achieved by shallowESN with the same total number of recurrent units. To evaluate the choice performed by the design algorithm, we re-trained the readout for each number of recurrent layers until the number of layers selected by the design algorithm is reached, i.e. 35 for Piano-midi.de and 36 for MuseData. Thereby, for DeepESN we considered an architecture with reservoirs composed by 200 recurrent units each, and with a number of layers chosen in the range $\{5, 10, 15, 20, 25, 30, selected\_layers\}$ where $selected\_layers$ equals 35 and 36 for Piano-midi.de and MuseData respectively. Accordingly, the number of total units considered for shallowESN is $\{1000, 2000, 3000, 4000, 5000, 6000, max\_units\}$ where max_units is 7000 and 7200 for Piano-midi.de and MuseData, respectively. For each configuration of the number of DeepESN layers (total number of units for the shallowESN), the networks were selected on the basis of the performance achieved on the validation set considering values of the hyper-parameters chosen from the ranges defined in Table 4.1.

Figures 5.6 and 5.7 show the validation FL-ACC obtained by DeepESN and shallowESN on Piano-midi.de and MuseData, respectively, for increasing number of layers and total number of recurrent units. For the sake of graphical comparison, in the same figures we also plotted the test performance achieved by RNN-RBM in [23] as a horizontal dashed line. As we can see from both Figures 5.6 and 5.7, DeepESN always performs better than shallowESN for every number of total recurrent units considered. Furthermore, the comparative plots clearly show that DeepESNs are able to reach (and even outperform) the performance achieved by shallowESNs with a much larger number of recurrent units, e.g. a DeepESN with only 1000 total recurrent units reaches a similar performance to what achieved by a shallowESN with a total reservoir size of 3000. Results in Figures 5.6 and 5.7 also point out that the choice of the design algorithm to select 35 layers for Piano-midi.de and 36 layers for MuseData is appropriate, especially in light of the saturation effect on the performance that can be appreciated in both cases after 30 layers. Moreover, looking at the relation between validation and test performance, we observed that DeepESN obtained validation FL-ACCs of 32.61 and 37.76 on Piano-midi.de and MuseData, respectively, with a deviation of 0.39 and 1.33 FL-ACCs from correspondent test errors. Such results highlight the effectiveness of the design choice also in what concerns the generalization error. Overall, in the studied cases, these results show the good ability of our approach in the automatic selection of the proper number of recurrent layers, also approaching challenging real-world tasks, reaching a good

performance able to outperform the state-of-the-art results and at the same time avoiding to build up a too complex model.

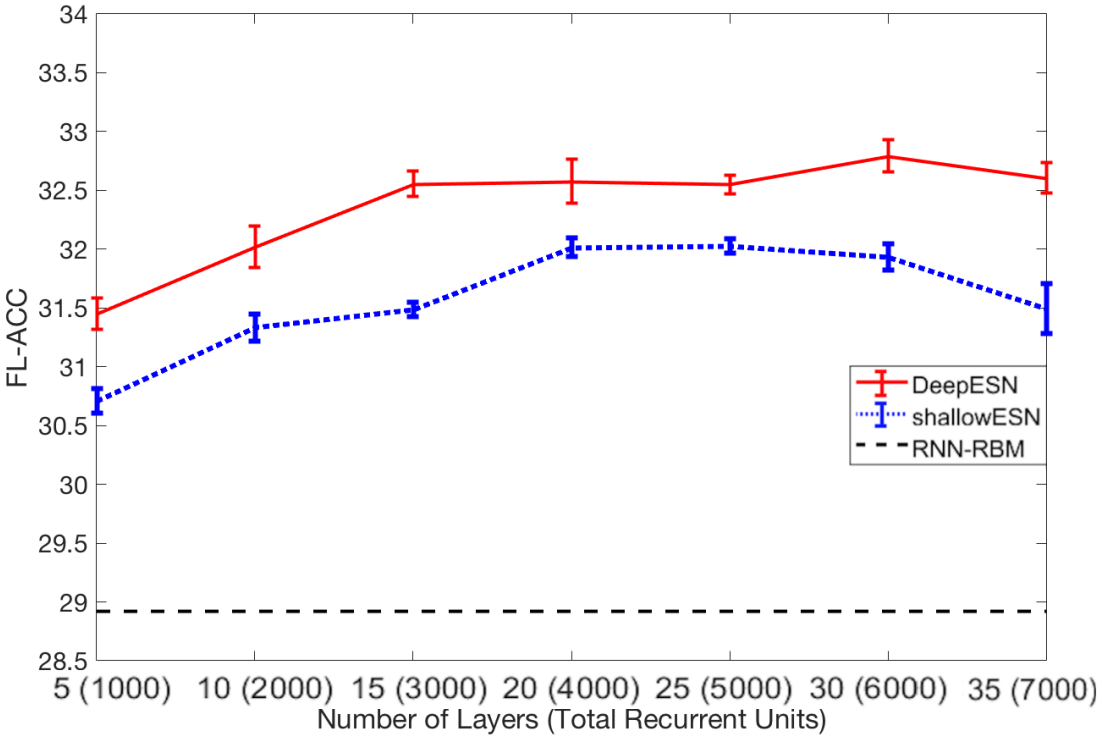


Figure 5.6: Comparison between the FL-ACCs (and standard deviations on reservoir guesses represented by vertical intervals) obtained on the validation set of the Piano-midi.de task by DeepESN and shallowESN, considering a number of recurrent layers in the range $5 - 35$ and, correspondingly, a total number of recurrent units in the range $1000 - 7000$ (results are obtained through model selection individually performed for each number of layers or total recurrent units for shallowESN). The dashed line at the bottom represents the test set performance achieved by RNN-RBM in [23].

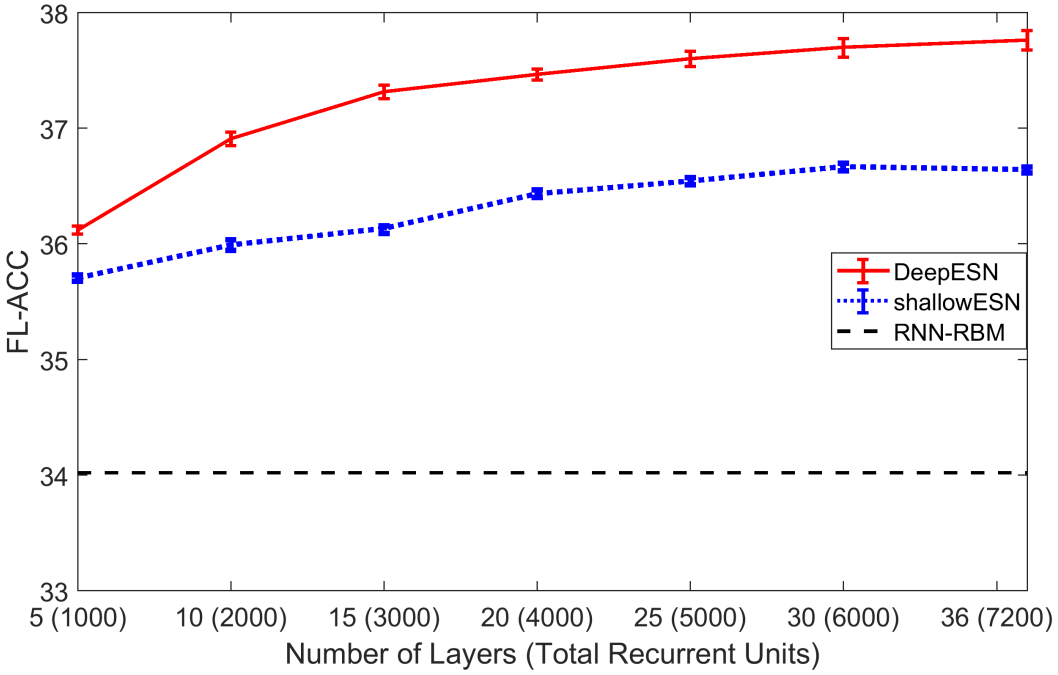


Figure 5.7: Comparison between the FL-ACCs (and standard deviations on reservoir guesses represented by vertical intervals) obtained on the validation set of the MuseData task by DeepESN and shallowESN, considering a number of recurrent layers in the range $5-36$ and, correspondingly, a total number of recurrent units in the range $1000-7200$ (results are obtained through model selection individually performed for each number of layers or total recurrent units for shallowESN). The dashed line at the bottom represents the test set performance achieved by RNN-RBM in [23].

5.3.2 Speech Recognition

Here, we consider the isolated spoken digit recognition task discussed in [201]. This is a widely used task in the RC context (see e.g. [200, 154, 155]), and it is characterized by multiple time-scales dependencies in high-dimensional sequences. The problem is modeled as a multi-class classification task, and it consists in recognizing ten (zero to nine) isolated spoken digits. Each digit is spoken 10 times by 5 different speakers. The 500 spoken digits are randomly split into training and test sets, each containing 250 sequences. As in works [200, 154, 155], the model selection is performed by 10-fold cross-validation on the training set, and the error is evaluated using the Word Error Rate (WER). As proposed in [200], the speech audio was preprocessed using a biological model of the human cochlea by [122],

resulting in a 77-channel cochleagram.

For what regards DeepESN, the model selection process is performed as in Section 5.3.1. Moreover, the performance obtained by DeepESN is compared with the one achieved by shallowESN model, considering a range of total recurrent units in $50 - 550$ (with a step of 50). For this task, the design algorithm selected a DeepESN with 11 recurrent layers.

Model	Test WER
DeepESN	**0.0028** (**0.0005**)
shallowESN	0.0530 (0.0318)
SRC [154]	0.0081 (0.0022)
CRJ [155]	0.0046 (0.0021)

Table 5.14: Test WER (and standard deviation on folds, shown in parentheses) obtained on the speech recognition task by DeepESN, shallowESN, SRC and CRJ.

Table 5.14 shows the test WER obtained by DeepESN designed with the proposed design algorithm, shallowESN, Simple Circular Reservoir (SCR) [154] and Circular Reservoir with Jumps (CRJ) [155]. The hyper parameters that obtained the best performance in validation set are $N_L = 11$, $N_R = 50$, $\rho = 0.7$, $scale_{\text{in}} = 10$, $a = 0.1$, $\sigma_{\text{IP}} = 0.1$ and $\lambda_r = 10^{-1}$ for DeepESN and $N_L = 1$, $N_R = 250$, $\rho = 1$, $scale_{\text{in}} = 10$, $a = 0.1$, $\sigma_{\text{IP}} = 0.1$ and $\lambda_r = 10^{-3}$ for shallowESN. As results show, on this task the difference between the performance of DeepESN and shallowESN is remarkable, with a gap of 0.0502 WER on the test set. Note that, the state-of-the-art approaches SRC and CRJ, already reached good results in this task with respect to shallowESN, achieving a test WER of 0.0081 and 0.0046, respectively. Moreover, such results are obtained by CRJ and SCR models using 300 recurrent units. We can note from Figure 5.8 that the error obtained by DeepESN using 300 recurrent units remains lower than the error achieved by such models considering the same number of free parameters.

Remarkably, the number of layers selected by the proposed design algorithm allows the DeepESN model to outperform the state of the art, reaching a test performance that is almost 4 and 2 times better than SRC and CRJ, respectively.

The effectiveness of the proposed approach is investigated by considering the results obtained by re-training the readout of DeepESNs with a progressively larger number of recurrent layers. Specifically, in this case the number of recurrent layers in the DeepESN architecture varied in the range $1 - 11$, while the total number of recurrent units was in the range $50 - 550$. The other hyper-parameters values were

chosen from the ranges in Table 4.1. Also in this case, the experimental analysis was conducted in comparison to the results achieved by shallowESN under the same experimental settings and with the same total number of recurrent units.

Figure 5.8 shows the validation WERs obtained by DeepESN and shallowESN by re-training the readout in correspondence of networks settings with a progressively larger number of layers (and recurrent units) in the architecture. In the same figure, we also indicated the test WERs achieved by the state-of-the-art models on the task, i.e. SRC and CRJ, as horizontal dash-dotted and dashed lines, respectively.

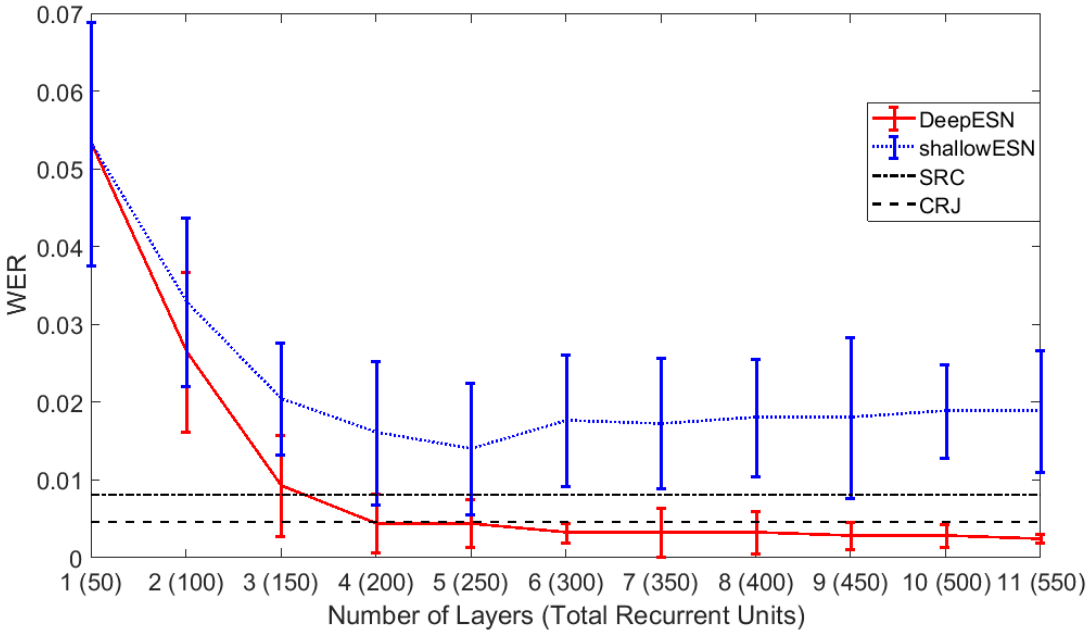


Figure 5.8: Comparison of WERs (and standard deviations on folds represented by vertical intervals) obtained on the validation set by DeepESN and shallowESN considering a number of recurrent layer in the range $1-11$ and a total number of recurrent units in the range $50-550$ on speech recognition task (results are obtained through model selection individually performed for each number of layers or total recurrent units for shallowESN). The horizontal dash-dotted and dashed lines respectively represent the test set performance achieved by SRC and CRJ in [155].

From Figure 5.8 we can see that DeepESN outperformed the shallowESN also on this task, for all the cases of total number of recurrent units considered. Moreover, we can see that DeepESNs required a smaller number of units to reach and outperform the performance of shallowESNs. For example, from the plot in Figure 5.8 we can see that DeepESNs with only 150 units in total were already able to beat the

results of shallowESNs with even up to more than three times larger reservoirs (i.e. up to 550 units). In general, we can see that for increasing number of layers the validation performance of the DeepESN continues to improve (i.e. the WER continues to decrease), with a saturation effect that can be observed also in this case. Finally, note that the DeepESN with 11 layers obtained a validation WER of 0.0024, with a corresponding test WER of 0.0028 (i.e. the validation-test deviation is $\approx 4 \times 10^{-4}$ WER) which suggests that the choice made by our proposed design method results effective also with respect to the generalization error.

5.3.3 Experimental Analysis of Depth on Real-world Tasks

In this section, we empirically evaluate how the spectral centroid of reservoir states, computed by Algorithm 3 on the layers of the DeepESN, varies with the depth of the stacked recurrent architecture on the considered real-world tasks (see Section 4.2.4 for methodological details). The aim is to assess the considered η value as well as to evaluate the effectiveness of the Algorithm 4 and the stop condition defined in Equation 4.1 on the considered real-world tasks.

Figure 5.9 shows the trend of the spectral centroid obtained from the state of each recurrent layer of the DeepESN, optimized on the speech recognition task and considering the value of $\eta = 0.01$. The red vertical line represents the number of selected layers. As in the case of FBC task (see Section 4.2.4), the layers

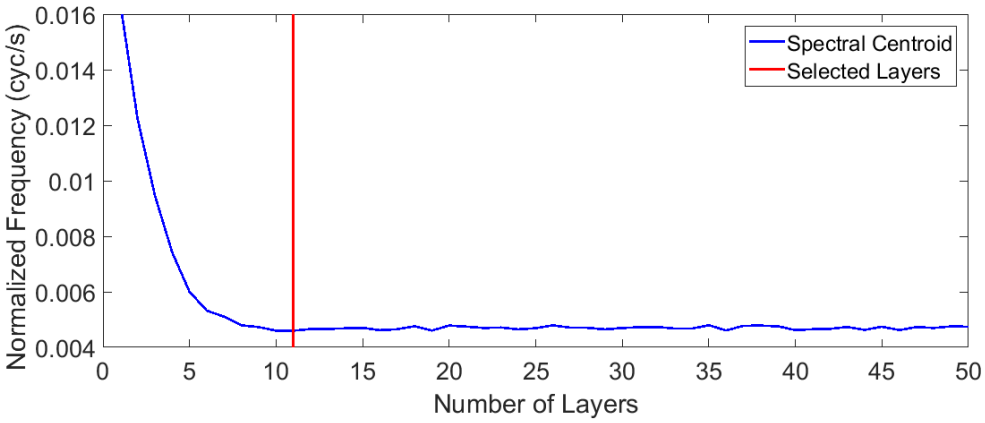


Figure 5.9: Spectral centroid computed on the state of DeepESN layers optimized on speech recognition task. The red vertical line indicates the number of layers selected by the design algorithm. Threshold η value is set to 0.01. Normalized Frequency is expressed in cycles per second (cyc/s).

progressively apply a low-pass filter to the signal. Moreover, also in this case, we can note that numerical FFT errors lead to small fluctuations of the spectral

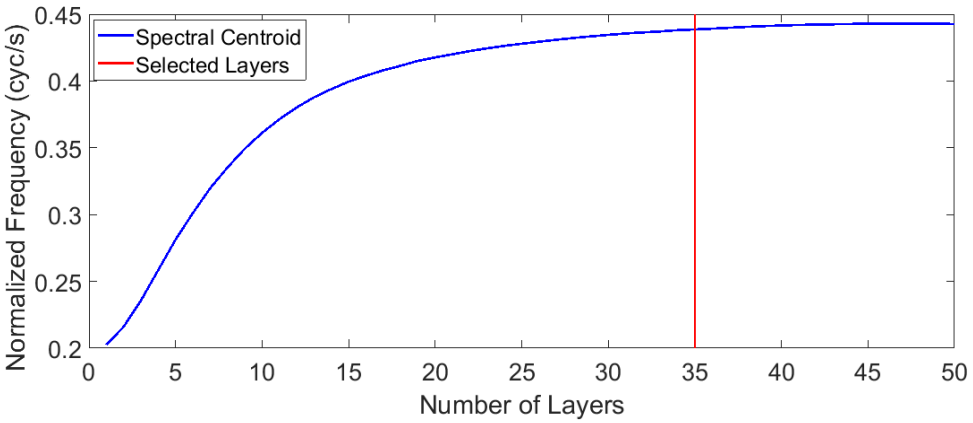


Figure 5.10: Spectral centroid computed on the state of DeepESN layers optimized on Piano-midi.de task. The red vertical line indicates the number of layers selected by the design algorithm. Threshold η value is set to 0.01. Normalized Frequency is expressed in cycles per second (cyc/s).

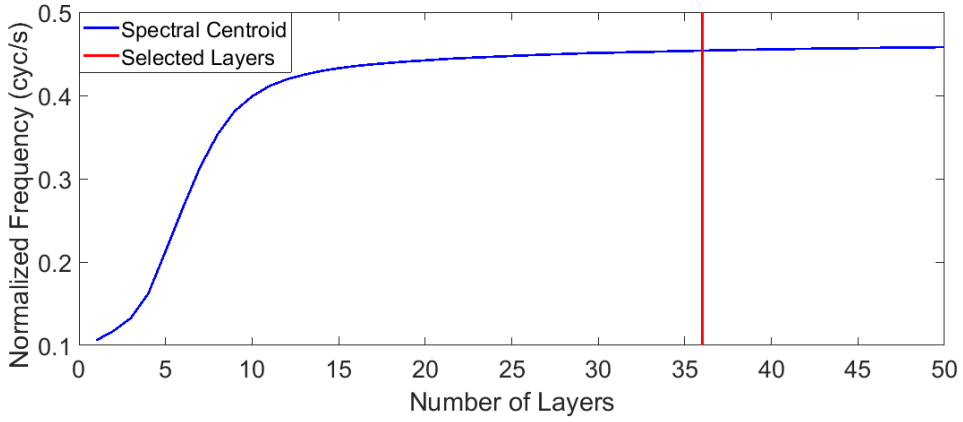


Figure 5.11: Spectral centroid computed on the state of DeepESN layers optimized on MuseData task. The red vertical line indicates the number of layers selected by the design algorithm. Threshold η value is set to 0.01. Normalized Frequency is expressed in cycles per second (cyc/s).

centroid after convergence. From this, we can observe that also for this task the value of $\eta = 0.01$ for the stop condition in Equation 4.1 is adequate to reach a point near the convergence.

For what regards the polyphonic music tasks (defined in Section 5.3.1), Figures 5.10 and 5.11 show the spectral centroid obtained from the state of each recurrent layer of DeepESN optimized on the Piano-midi.de and MuseData tasks respectively. Note that, differently from the cases of FBC (in Figure 4.4) and speech recognition (in Figure 5.9) tasks, the spectral centroid shifts progressively on high frequencies. This can depend on many factors such as leaky integration, spectral radius of $\hat{\mathbf{W}}$, input scaling and IP adaptation. Please note that, although these empirical results on high-pass filtering are very interesting in themselves, the analytical study of this aspect in relation to the hyper-parameterization of the reservoirs is out of the scope of this work. However, even in this case, the filtering effect becomes progressively negligible in determining the convergence of the spectral centroid. Moreover, we can note that the trend of the spectral centroid is smooth and the convergence is asymptotic. This is a reason why the η value should be greater than 0. However, on these tasks the spectral centroid (Figures 5.10 and 5.11) and the performance (Figures 5.6 and 5.7) tend to saturate, thus leading to small differences for both the number of selected layers and the performance obtained if η is set to values smaller than 0.01 (see considerations regarding the performance obtained on these tasks in Section 5.3.1). Overall, we can conclude that on all considered cases, the value of $\eta = 0.01$ resulted to be an empirically adequate threshold for the stop condition (Equation 4.1) of the proposed design algorithm. Nevertheless, it is worth to observe that the methodology assumed by looking at the trend of the spectral centroid over the layers, similarly to the plots shown in this section, can be used to fix a value of η tailored to different tasks.

5.4 Comparison Between DeepESNs and Gated RNNs

In this section we investigate different approaches to RNN modeling (i.e., untrained stacked layers and fully-trained gated architectures), through an experimental comparison between RC and fully-trained RNNs on challenging real-world prediction tasks characterized by multivariate time series. In particular, we perform a comparison between DeepESN, LSTM and GRU models (see Section 2.3

for RNN architectures description) on polyphonic music tasks described in Section 5.3.1. For the sake of completeness, in this comparison we take into account all 4 polyphonic music tasks introduced in [23], namely Piano-midi.de, MuseData, JSBchorales[8] and Nottingham[9]. These applications concern next-step prediction tasks in which data is composed by 88-, 82-, 52- and 58- dimensional sequences for Piano-midi.de, MuseData, JSBchorales and Nottingham tasks, respectively (See Section 5.3.1 for further details). Since these datasets are characterized by sequences with high-dimensionality and complex temporal sequences, these challenging tasks are particularly suitable for RNNs evaluation [13]. Moreover, we consider ESN and simple RNN (Simple Recurrent Network - SRN) as baseline approaches for DeepESN and gated RNNs, respectively. The models are evaluated in terms of predictive accuracy and computation efficiency.

In a context in which the model design is difficult, especially for fully-trained RNNs, we would provide a first glimpse in the experimental comparison between different state-of-the-art recurrent models on multivariate time-series prediction tasks which still lacks in literature.

Concerning DeepESN and ESN approaches, we considered reservoirs initialized with 1% of connectivity. Moreover, we performed a model selection on the major hyper parameters considering spectral radius ρ and leaky integrator a values in $\{0.1, 0.3, 0.5, 0.7, 0.9, 1.0\}$, and input scaling σ values in $\{0.5, 1.5, 2.5\}$. Training of the readout was performed through ridge regression [99, 121] with regularization coefficient λ_r in $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$. Moreover, based on the results of the design analysis in [62] on polyphonic music tasks, we set up DeepESN with $N_L = 30$ layers composed by $N_R = 200$ units, and ESN with $N_R = 6000$ recurrent units. We used an IP adaptation configured as in [61, 62] with a standard deviation of $\sigma_{\text{IP}} = 0.1$.

For what regards fully trained RNNs, we used the Adam learning algorithm [106] with a maximum of 2000 epochs. In order to regularize the learning process, we applied dropout methods, a clipping gradient with a value of 5 and an early stopping with a patience value of 30. Then, we performed a model selection considering learning rate values in $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$ and dropout values in $\{0.1, 0.2, 0.3, 0.4, 0.5\}$.

Since randomized and fully-trained RNNs implement different learning approaches, it is difficult to set up a fair experimental comparison between them. However,

[8] A corpus of 382 harmonized chorales by J. S. Bach with the split of Allan Williams (2005)
[9] A collection of 1200 folk tunes ifdo.ca/~seymour/nottingham/nottingham.html

Model	total recurrent units	free parameters	test ACC	computation time
Piano-midi.de				
DeepESN	6000	540088	**33.33 (0.11) %**	**386**
ESN	6000	540088	30.43 (0.06) %	748
SRN	652	540596	29.48 (0.35) %	3185
LSTM	316	539816	28.98 (2.93) %	2333
GRU	369	539566	31.38 (0.21) %	2821
MuseData				
DeepESN	6000	504082	**36.32 (0.06) %**	**789**
ESN	6000	504082	35.95 (0.04) %	997
SRN	632	503786	34.02 (0.28) %	8825
LSTM	307	504176	34.71 (1.17) %	18274
GRU	358	503072	35.89 (0.17) %	18104
JSBchorales				
DeepESN	6000	324052	**30.82 (0.12) %**	**83**
ESN	6000	324052	29.14 (0.09) %	140
SRN	519	323908	29.68 (0.17) %	341
LSTM	254	325172	29.80 (0.38) %	532
GRU	295	323372	29.63 (0.64) %	230
Nottingham				
DeepESN	6000	360058	69.43 (0.05) %	**677**
ESN	6000	360058	69.12 (0.08) %	1473
SRN	545	360848	65.89 (0.49) %	2252
LSTM	266	361286	70.00 (0.24) %	26175
GRU	309	359116	**71.50 (0.77)** %	11844

Table 5.15: free parameters and test ACC achieved by DeepESN, SRN, LSTM and GRU. Computation time represents the seconds to complete training and test.

we faced these difficulties by considering a comparable number of free parameters for all the models. The number of recurrent units and free parameters considered in the models is shown in the second and third columns of Tab. 5.15. Each model is individually selected on the validation sets through a grid search on hyper-parameters ranges. We independently generated 5 guesses for each network hyper-parametrization (for random initialization), and averaged the results over such guesses.

In accordance with the different characteristics of the considered training approaches (direct methods for RC and iterative methods for fully-trained models) we preferred the most efficient method in all the considered cases. Accordingly, we used a MATLAB implementation for DeepESN and ESN models, and a Keras implementation for fully-trained RNNs. We measured the time in seconds spent by models in training and test procedures, performing experiments on a CPU "Intel Xeon E5, 1.80GHz, 16 cores" in the case of RC approaches, and on a GPU "Tesla P100 PCIe 16GB" in the case of fully-trained RNNs, with the same aim to give the

best resource to each of them.

Tab. 5.15 shows the number of recurrent units, the number of free parameters, the predictive accuracy and the computation time (in seconds) achieved by DeepESN, ESN, SRN, LSTM and GRU models. For what regards the comparison between RC approaches in terms of predictive performance, results indicate that DeepESN outperformed ESN with an accuracy improvement of 2.90%, 0.37%, 1.68% and 0.31% on Piano-midi.de, MuseData, JSBchorales and Nottingha tasks, respectively. Concerning the comparison between fully-trained RNNs, GRU obtained a similar accuracy to SRN and LSTM models on JSBchorales task and it outperformed them on Piano-midi.de, MuseData and Nottingham tasks.

The efficiency assessments show that DeepESN requires about less that one order of magnitude of computation time with respect to fully-trained RNNs, boosting the already striking efficiency of standard ESN models. Moreover, while ESN benefits in terms of efficiency only by exploiting the sparsity of reservoirs (with 1% of connectivity), in the case of DeepESN the benefit is intrinsically due to the architectural constraints involved by layering [62] (and are obtained also with fully-connected layers).

Overall, the DeepESN model outperformed all the other approaches on 3 out of 4 tasks, resulting extremely more efficient with respect to fully-trained RNNs. This kind of comparisons in complex temporal tasks, that is practically absent in literature especially for what regards efficiency aspects, offered the opportunity to assess efficient alternative models (ESN and DeepESN in particular) to typical RNN approaches (LSTM and GRU). Moreover, we assessed also the effectiveness of layering in deep recurrent architectures with a large number of layers (i.e., 30).

Concerning fully-trained RNNs, GRU outperformed the other gated RNNs on 3 out of 4 tasks and it was more efficient than LSTM in most cases. The effectiveness of GRU approaches found in our experiments is in line with the literature that deals with the design of adaptive gates in recurrent architectures.

For what regards randomized RNNs, the results show that DeepESN is able to outperform ESN in terms of prediction accuracy and efficiency on all tasks. Interestingly, this highlights that the layering aspect allows us to improve the effectiveness of RC approaches on multiple time-scales processing. Overall, the DeepESN model outperformed other approaches in terms of prediction accuracy on 3 out of 4 tasks. Finally, DeepESN required much less time in computation time with respect to the others models resulting in an extremely efficient model able to compete with the state of the art on challenging time-series tasks.

More in general, it is interesting to highlight the gain in the prediction accuracy showed by the multiple time-scales processing capability obtained by layering in deep RC models and by using adaptive gates in fully-trained RNNs in comparison to the respective baselines (ESN and SRN, respectively). Also, it is particularly interesting to note the comparison between models with the capability to learn multiple time-scales dynamics (LSTM and GRU) and models showing an intrinsic capability to develop such kind of hierarchical temporal representations (DeepESN), which was completely lacking in literature.

In addition to provide insights on such general issues, this experimental assessment would contribute to show a practical way to efficiently approach the design of learning models in the scenario of deep RNN, extending the set of tools available to the users for complex time-series tasks. Indeed, the first empirical results provided in these experiments seem to indicate that some classes of models are sometimes uncritically adopted, i.e. despite their cost, guided by the natural popularity due to their software availability (GRU, LSTM). The same diffusion of software tools deserve more effort on the side of the other models (DeepESN class), although the first instances are already available[10].

The superiority of DeepESN over fully-trained RNNs highlighted in these experimental assessments open the way for future works concerning the comparison between DeepESN and gated RNNs in tasks where fully-trained RNNs results dominant such as natural language processing [132], continuous speech recognition [73] and encoder-decoder mechanisms [9, 186].

[10]DeepESN implementations are made publicly available for download both in MATLAB (see https://it.mathworks.com/matlabcentral/fileexchange/69402-deepesn) and in Python (see https://github.com/lucapedrelli/DeepESN).

Conclusions

In this thesis we introduced a novel class of randomized deep RNNs explicitly extending the RC framework to Deep Learning paradigm. The main research issues addressed in our studies concern the investigation of the merits of layering in deep recurrent architectures, the analysis and design of deep models through quantitative and qualitative measures and the development of competitive and efficient deep RNNs for real-world tasks characterized by multiple time-scales dynamics data. The recourse to RC paradigm allowed us to analyze deep recurrent architectures separately from learning algorithms aspects and at the same time to exploit the efficiency that characterizes RC models.

We started with the experimental analysis to assess the intrinsic ability of layering in developing a multiple time-scales differentiation among the deep RNN architecture. In order to study and analyze the temporal features developed by layering in deep RNNs, we introduced the DeepESN model characterized by a hierarchy of non-linear recurrent layers and a linear output layer. The first layer is fed by the external input, while the successive layers are fed by previous ones. The recurrent layers are randomly initialized, whereas the output layer is trained by using direct methods. The output of the network is computed through a weighted sum of the states coming from all recurrent layers in the architecture. In this way, the training phase allows the model to weights the contributions of the state dynamics involved in the architecture. Moreover, we introduced a pre-training algorithm for deep RNN based on IP algorithm. Thereby, the IP algorithm is incrementally applied through the recurrent layers in unsupervised manner.

The importance of layering is assessed by comparing DeepESN with other specific architectural variants used as baseline (i.e., DeepESN-IA, GroupedESN and shallowESN). In particular, DeepESN-IA is a DeepESN variant in which the external input is provided to every hidden layer. Thereby, in this case, the distance from the external input is the same for every layer, whereas in DeepESN higher layers are at

increasing distances from the input. Another architectural baseline considered is GroupedESN in which layers are fed only by the input and are not organized in a stack. Note that in this architectural case, although the recurrent layers can provide decoupled multiple time-scales dynamics, the temporal features are not developed by the interplay between layers (as in DeepESN), since the distance between the external input and the network's output is the same as in shallow RNNs. As a further architectural baseline, we also considered the shallowESN, with the same number of total reservoir units of deep RC counterpart. Overall, we compared DeepESN with other architectural baselines in terms of multiple time-scales differentiation and memory capacity varying RC parameters and applying the IP algorithm.

The empirical analysis on the considered model topologies show that the constrained characterization of DeepESN architectures, obtained by imposing the hierarchical organization of layers, is crucial to enhance the filtering effect between the input information and the output layer and to inherently improve the temporal representation in terms of multiple time-scales differentiation, memory capacity and quantity of entropy. This characterization is obtained even in the case in which all the recurrent layers share the same hyper parameters. Moreover, we obtained an enhancement of temporal differentiation developed by the architecture by varying the RC parameters among the layers. Another relevant improvement in the richness of temporal representation is achieved by applying progressively the IP algorithm trough the layers. A further advantage obtained by the constraint characterization of DeepESN is the efficiency of the state computation. Indeed, by comparison with shallowESN, the DeepESN architecture with the same number of units has significantly fewer recurrent weights.

In order to take a step forward in the study of temporal features developed by a layered RNN, we introduced the L-DeepESN architecture as a variant of DeepESN which implements linear activation functions. Through L-DeepESN, we performed the frequency analysis of the temporal features developed by a stack of recurrent layers. The use of linear activations allowed us to study the characteristics of the frequency spectrum avoiding harmonic distortion effects in the frequency analysis of network's dynamics. Moreover, we provided an algebraic characterization of the layered architecture to highlight the filtering effect of layering.

The analysis conducted with L-DeepESN highlighted an ordered structure of frequency components intrinsically developed by the architecture. Indeed, higher frequency components are progressively filtered in higher layers. The filtering effect emerges also from the proposed algebraic characterization for layered architectures.

Interestingly, we found that layering in recurrent architectures is intrinsically able to develop hierarchical and distributed temporal features even without the use of non-linear transformation (and without the training recurrent weights).

After the study of layering characterization, we introduced a novel design approach to determine the number of layers in a DeepESN based on the frequency analysis of the network's state. Accordingly, we developed an efficient tool to analyze the filtering effect progressively performed in the depth of a deep RNN. On the experimental side, we have first refined the approach on a controlled scenario characterized by predefined multiple time-scale dynamics, then, we assessed the design approach on real-world tasks in the area of speech processing and polyphonic music.

Through the design tools introduced in this thesis, we analyzed the filtering effect among the recurrent layers of a DeepESN. On the one hand, this approach allowed us to efficiently design deep RNNs avoiding to perform the training algorithm for each possible number of recurrent layers. On the other hand, differently to other literature approaches, our method achieved competitive results exploiting the temporal representation developed by many recurrent layers (>30). Moreover, the design tools allowed us to find another interesting result in the study of the filtering effect in recurrent layers. Indeed, contrary to other cases in literature in which recurrent layers act as a low-pass filter, our empirical analysis showed that it is also possible to produce an high-pass filtering effect by using the IP algorithm.

For what regards the experimental assessments on real-world tasks, we introduced novel applications based on RC paradigm and we evaluated the DeepESN model on all tasks for different purposes.

First, we introduced a novel learning system for human balance assessment through the RC paradigm. The system is based on estimating the berg balance scale score from the stream of sensor data gathered by an electronic balance board. In order to develop the system, we collected a new dataset gathering the data from the execution of exercises on the electronic balance board by volunteer users. We compared the RC approach with different basic learning models. The results highlighted the ability of the RC approach in capture relevant information from the whole temporal signal, allowing to effectively improve state-of-the-art results in BBS estimation. For the aim of this thesis, we also compared the DeepESN model with the shallow counterpart on the baseline exercise task in order to assess the advantage of layering in terms of performance. The DeepESN model obtained a performance improvement with respect to the shallow counterpart on the balance

assessment task.

Another medical application developed concerns the diagnosis of Parkinson's disease through DeepESN model. The model is evaluated on a public dataset of spiral tests and compared with the shallow counterpart. The experiment assessment shows that the DeepESN model outperforms the shallowESN model and the other literature approaches in this kind of datasets. The results presented in this thesis suggest that the proposed approaches are suitable for medical applications composed by noisy signals characterized by multiple time-scales dynamics.

Finally, we assessed the DeepESN model on challenging real-world tasks in the field of sequence classification and time-series prediction. We analyzed and evaluated our design approach for DeepESN on speech recognition and polyphonic music tasks. In particular, since polyphonic music tasks are characterized by high-dimensional time series, heterogeneous sequences and sparse vector representations, they represent good real-world benchmarks for time-series prediction. Therefore, on 4 polyphonic music tasks, we compared the DeepESN model with typical state-of-the-art RNNs in time-series processing namely SRN, GRU and LSTM.

The experimental assessments showed that DeepESN outperforms the shallow counterpart on all considered tasks. Moreover, DeepESN resulted competitive with the other RC approaches and the fully trained RNNs of the state of the art. Interestingly, DeepESN outperformed all the considered gated RNNs on 3 out of 4 time-series prediction tasks based on 4 polyphonic music datasets obtaining better training time performances.

Overall, the analysis conducted on DeepESN model highlighted that the layering aspect in deep RNNs intrinsically enables to develop rich representations of temporal information characterized by multiple time-scales dynamics. These results are supported by quantitative and qualitative experimental evaluations. Indeed, the analysis showed that layering in deep recurrent architectures improves the richness of temporal representation in terms of time-scales differentiation, entropy, memory capacity and performance. Moreover, the filtering effect operated by hierarchy of layers intrinsically develops an ordered structure of frequency components among the layers. Through quantitative and qualitative measures, we introduced a design tool that allowed us to select the number of layers of deep recurrent architectures avoiding expensive trial and error procedures. The combination of randomized RNNs and Deep Learning paradigm allows DeepESN architectures to exploit the intrinsic characterization of layering, developing rich temporal representations, without the training of hidden layers. Hence, this provide extremely efficient solutions

for deep recurrent architecture avoiding the typical difficulties in training deep models and RNNs. Finally, the empirical analysis on challenging real-world tasks characterized by multiple time-scales dynamics highlighted that DeepESN models are extremely more efficient than fully trained RNNs, they are also competitive in terms of accuracy with the state-of-the-art approaches.

The studies and the approaches presented in this thesis open the way for future works. Concerning filtering aspects, the qualitative analysis on the considered tasks empirically showed that recurrent layers of a DeepESN can act as a low-pass filter but also as a high-pass filter. Therefore, further investigations could concern analytical/theoretical studies on the characterization of the filtering effect operated by recurrent layers in deep RNNs. In particular, we could analytically study which are the hyper parameters and parameters factors that determine the kind of filtering effect.

The design approach for randomized deep RNNs proposed in this thesis could also be considered to study the architectural initialization of fully trained multi-layered architectures with stochastic gradient descent approaches. This is particularly interesting in consideration of the difficulties that are typically encountered in training deep networks. Indeed, initialization approaches based on pre-training analysis can influence the efficiency and the stability of gradient based algorithms in deep nonlinear networks.

Giving the good efficiency provided by our approaches, the evaluation of DeepESN on other relevant applications characterized by bigger datasets in the field of natural language processing and computer vision deserves to be investigated. Moreover, in addition to the comparison presented in this thesis, it would be interesting to perform further comparisons between DeepESN, fully trained deep RNNs and other recent literature approaches in RNN field.

Another line of research could regard the design and the implementation of other advanced methods for fully trained RNNs such as attention and encoder-decoder mechanisms [9, 186] in DeepESNs approaches. The difficulty in this case is to combine direct and iterative learning algorithms with randomized RNNs.

Finally, in this thesis we presented suitable approaches for sequence modelling. Further works could generalize this methods in order to model more general structures such as trees and graphs [46, 57]. This would allow us to evaluate the proposed analysis and design approaches on a wide range of application fields characterized by structured information.

Bibliography

[1] D. Albers, J. Sprott, and W. Dechert. "Dynamical behavior of artificial neural networks with random weights". In: *Intelligent Engineering Systems Through Artificial Neural Networks* 6 (1996), pp. 17–22.

[2] S. Anderson, J.W.L. Merrill, and R. Port. *Dynamic speech categorization with recurrent networks*. Indiana University, Computer Science Department, 1988.

[3] P. Angelov and A. Sperduti. "Challenges in Deep Learning". In: *Proceedings of the 24th European Symposium on Artificial Neural Networks (ESANN)*. i6doc.com. 2016, pp. 489–495.

[4] D. Bacciu and A. Bongiorno. "Concentric ESN: Assessing the Effect of Modularity in Cycle Reservoirs". In: *CoRR* abs/1805.09244 (2018). arXiv: 1805.09244. URL: http://arxiv.org/abs/1805.09244.

[5] D. Bacciu, S. Chessa, C. Gallicchio, A. Micheli, E. Ferro, L. Fortunati, F. Palumbo, O. Parodi, F. Vozzi, S. Hanke, J. Kropf, and K. Kreiner. "Smart Environments and Context-Awareness for Lifestyle Management in a Healthy Active Ageing Framework". In: *Progress in Artificial Intelligence*. Springer, 2015, pp. 54–66.

[6] D. Bacciu, S. Chessa, C. Gallicchio, A. Micheli, L. Pedrelli, E. Ferro, L. Fortunati, D. La Rosa, F. Palumbo, F. Vozzi, and O. Parodi. "A learning system for automatic Berg Balance Scale score estimation". In: *Engineering Applications of Artificial Intelligence* 66 (2017), pp. 60–74. DOI: 10.1016/j.engappai.2017.08.018.

[7] D. Bacciu, C. Gallicchio, A. Micheli, S. Chessa, and P. Barsocchi. "Predicting user movements in heterogeneous indoor environments by reservoir computing". In: *Proceedings of the IJCAI workshop on space, time and ambient intelligence (STAMI)*. 2011, pp. 1–6.

[8] P. Badura and E. Pietka. "Automatic Berg Balance Scale assessment system based on accelerometric signals". In: *Biomedical Signal Processing and Control* 24 (2016), pp. 114–119.

[9] D. Bahdanau, K. Cho, and Y. Bengio. "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (2014).

[10] *Balance Assessment Dataset and Supplementary Material*. http://www.di.unipi.it/groups/ciml/Data/balance.html. 2016.

[11] M. Bay, A. F Ehmann, and J. S. Downie. "Evaluation of Multiple-F0 Estimation and Tracking Systems." In: *10th International Society for Music Information Retrieval Conference (ISMIR)*. 2009.

[12] J. Beirlant, E. J. Dudewicz, L. Györfi, and E. C. Van der Meulen. "Nonparametric entropy estimation: An overview". In: *International Journal of Mathematical and Statistical Sciences* 6.1 (1997), pp. 17–39.

[13] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu. "Advances in optimizing recurrent networks". In: *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE. 2013, pp. 8624–8628.

[14] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. "Greedy layer-wise training of deep networks". In: *Advances in neural information processing systems*. 2007, pp. 153–160.

[15] Y. Bengio, P. Simard, and P. Frasconi. "Learning long-term dependencies with gradient descent is difficult". In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166.

[16] N. Bertschinger and T. Natschläger. "Real-time computation at the edge of chaos in recurrent neural networks". In: *Neural computation* 16.7 (2004), pp. 1413–1436.

[17] F. M. Bianchi, M. Kampffmeyer, E. Maiorino, and R. Jenssen. "Temporal overdrive recurrent neural network". In: *Neural Networks (IJCNN), 2017 International Joint Conference on*. IEEE. 2017, pp. 4275–4282.

[18] H. Block. "The perceptron: A model for brain functioning. i". In: *Reviews of Modern Physics* 34.1 (1962), p. 123.

[19] L. Blum and N. Korner-Bitensky. "Usefulness of the Berg Balance Scale in stroke rehabilitation: a systematic review". In: *Physical therapy* 88.5 (2008), pp. 559–566.

[20] J. Boedecker, O. Obst, N. M. Mayer, and M. Asada. “Studies on reservoir initialization and dynamics shaping in echo state networks.” In: *ESANN*. 2009.

[21] L. Bottou. “Online Algorithms and Stochastic Approximations”. In: *Online Learning and Neural Networks*. Ed. by David Saad. revised, oct 2012. Cambridge, UK: Cambridge University Press, 1998. URL: http://leon.bottou.org/papers/bottou-98x.

[22] L. Bottou and O. Bousquet. “The Tradeoffs of Large Scale Learning”. In: *Advances in Neural Information Processing Systems*. Ed. by J.C. Platt, D. Koller, Y. Singer, and S. Roweis. Vol. 20. NIPS Foundation (http://books.nips.cc), 2008, pp. 161–168. URL: http://leon.bottou.org/papers/bottou-bousquet-2008.

[23] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent. “Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription”. In: *arXiv preprint arXiv:1206.6392* (2012).

[24] D. S. Broomhead and D. Lowe. *Radial basis functions, multi-variable functional interpolation and adaptive networks*. Tech. rep. DTIC Document, 1988.

[25] J. Browne and N. O’Hare. “Review of the different methods for assessing standing balance”. In: *Physiotherapy* 87.9 (2001), pp. 489–495.

[26] D. Cattaneo, A. Regola, and M. Meotti. “Validity of six balance disorders scales in persons with multiple sclerosis”. In: *Disability and rehabilitation* 28.12 (2006), pp. 789–795.

[27] M. Čerňanskỳ and P. Tiňo. “Predictive modeling with echo state networks”. In: *International Conference on Artificial Neural Networks*. Springer. 2008, pp. 778–787.

[28] Q. Chen, L. Shi, J. Na, X. Ren, and Y. Nan. “Adaptive echo state network control for a class of pure-feedback systems with input and output constraints”. In: *Neurocomputing* 275 (2018), pp. 1370–1382.

[29] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. “Learning phrase representations using RNN encoder-decoder for statistical machine translation”. In: *arXiv preprint arXiv:1406.1078* (2014).

[30] J. Chung, S. Ahn, and Y. Bengio. “Hierarchical multiscale recurrent neural networks”. In: *arXiv preprint arXiv:1609.01704* (2016).

[31] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. “Empirical evaluation of gated recurrent neural networks on sequence modeling”. In: *arXiv preprint arXiv:1412.3555* (2014).

[32] R. A. Clark, A. L. Bryant, Y. Pua, P. McCrory, K. Bennell, and M. Hunt. “Validity and reliability of the Nintendo Wii Balance Board for assessment of standing balance”. In: *Gait and Posture* 31.3 (2010), pp. 307–310. ISSN: 09666362. DOI: 10.1016/j.gaitpost.2009.11.012.

[33] A. Conneau, H. Schwenk, Loïc Barrault, and Yann Lecun. “Very deep convolutional networks for text classification”. In: *arXiv preprint arXiv:1606.01781* (2016).

[34] M. Conradsson, L. Lundin-Olsson, N. Lindelöf, H. Littbrand, L. Malmqvist, Y. Gustafson, and E. Rosendahl. “Berg balance scale: intrarater test-retest reliability among older people dependent in activities of daily living and living in residential care facilities”. In: *Physical Therapy* 87.9 (2007), pp. 1155–1163.

[35] E. Crisostomi, C. Gallicchio, A. Micheli, M. Raugi, and M. Tucci. “Prediction of the Italian electricity price for smart grid applications”. In: *Neurocomputing* 170 (2015), pp. 286–295.

[36] Y. Le Cun, B. Boser, J.S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. “Handwritten digit recognition with a back-propagation network”. In: *Advances in neural information processing systems*. Citeseer. 1990.

[37] G. Cybenko. “Approximation by superpositions of a sigmoidal function”. In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[38] J. Dan, W. Guo, W. Shi, B. Fang, and T. Zhang. “Deterministic echo state networks based stock price forecasting”. In: *Abstract and Applied Analysis*. Vol. 2014. Hindawi. 2014.

[39] J. E. Deutsch, M. Borbely, J. Filler, K. Huhn, and P. Guarrera-Bowlby. “Use of a low-cost, commercially available gaming console (Wii) for rehabilitation of an adolescent with cerebral palsy”. In: *Physical therapy* 88.10 (2008), pp. 1196–1207.

[40] M. Dragone, C. Gallicchio, R. Guzman, and A. Micheli. “Rss-based robot localization in critical environments using reservoir computing”. In: *Proceedings of the 24th European Symposium on Artificial Neural Networks (ESANN)*. 2016, pp. 71–76.

[41] J. L. Elman. “Distributed representations, simple recurrent networks, and grammatical structure”. In: *Machine learning* 7.2-3 (1991), pp. 195–225.

[42] J. L. Elman. “Finding structure in time”. In: *Cognitive science* 14.2 (1990), pp. 179–211.

[43] D. Erhan, Y. Bengio, A. Courville, P. Manzagol, P. Vincent, and S. Bengio. “Why does unsupervised pre-training help deep learning?” In: *Journal of Machine Learning Research* 11.Feb (2010), pp. 625–660.

[44] I. Farkaš, R. Bosák, and P. Gergel’. “Computational analysis of memory capacity in echo state networks”. In: *Neural Networks* 83 (2016), pp. 109–120.

[45] I. Farkaš and P. Gergel’. “Maximizing memory capacity of echo state networks with orthogonalized reservoirs”. In: *Neural Networks (IJCNN), 2017 International Joint Conference on*. IEEE. 2017, pp. 2437–2442.

[46] P. Frasconi, M. Gori, and A. Sperduti. “A general framework for adaptive processing of data structures”. In: *IEEE transactions on Neural Networks* 9.5 (1998), pp. 768–786.

[47] M. Frigo and S. G. Johnson. “FFTW: An adaptive software architecture for the FFT”. In: *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*. Vol. 3. IEEE. 1998, pp. 1381–1384.

[48] S. Gallant. “Random Cells: an idea whose time has come and gone... and come again?” In: *IEEE International Conference on Neural Networks*. Vol. 671. IEEE. 1987.

[49] C. Gallicchio. “Short-term memory of Deep RNN”. In: *Proceedings of the 26th European Symposium on Artificial Neural Networks (ESANN)*. i6doc.com. 2018, pp. 633–638.

[50] C. Gallicchio, J.D. Martin-Guerrero, A. Micheli, and E. Soria-Olivas. “Randomized Machine Learning Approaches: Recent Developments and Challenges”. In: *Proceedings of the 25th European Symposium on Artificial Neural Networks (ESANN)*. i6doc.com. 2017, pp. 77–86.

[51] C. Gallicchio and A. Micheli. "A Markovian characterization of redundancy in echo state networks by PCA." In: *ESANN*. Citeseer. 2010.

[52] C. Gallicchio and A. Micheli. "Architectural and markovian factors of echo state networks". In: *Neural Networks* 24.5 (2011), pp. 440–456.

[53] C. Gallicchio and A. Micheli. "Deep Echo State Network (DeepESN): A Brief Survey". In: (2017). arXiv:1712.04323.

[54] C. Gallicchio and A. Micheli. "Deep Reservoir Computing: A Critical Analysis". In: *Proceedings of the 24th European Symposium on Artificial Neural Networks (ESANN)*. i6doc.com. 2016, pp. 497–502.

[55] C. Gallicchio and A. Micheli. "Echo State Property of Deep Reservoir Computing Networks". In: *Cognitive Computation* 9.3 (2017), pp. 337–350. DOI: 10.1007/s12559-017-9461-9.

[56] C. Gallicchio and A. Micheli. "Experimental analysis of deep echo state networks for ambient assisted living". In: *Proceedings of the 3rd Workshop on Artificial Intelligence for Ambient Assisted Living (AI*AAL 2017), co-located with the 16th International Conference of the Italian Association for Artificial Intelligence (AI*IA 2017)*. 2018.

[57] C. Gallicchio and A. Micheli. "Tree echo state networks". In: *Neurocomputing* 101 (2013), pp. 319–337.

[58] C. Gallicchio and A. Micheli. "Why layering in Recurrent Neural Networks? A DeepESN Survey". In: *Proceedings of the 2018 IEEE International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2018, pp. 1800–1807.

[59] C. Gallicchio, A. Micheli, and L. Pedrelli. "Comparison between DeepESNs and gated RNNs on multivariate time-series prediction". In: *Proceedings of the 27th European Symposium on Artificial Neural Networks (ESANN)* (2019). (accepted).

[60] C. Gallicchio, A. Micheli, and L. Pedrelli. "Deep Echo State Networks for Diagnosis of Parkinson's Disease". In: *Proceedings of the 26th European Symposium on Artificial Neural Networks (ESANN)*. i6doc.com. 2018, pp. 397–402.

[61] C. Gallicchio, A. Micheli, and L. Pedrelli. "Deep reservoir computing: a critical experimental analysis". In: *Neurocomputing* 268 (2017), pp. 87–99. DOI: 10.1016/j.neucom.2016.12.089.

[62] C. Gallicchio, A. Micheli, and L. Pedrelli. "Design of deep echo state networks". In: *Neural Networks* 108 (2018), pp. 33 –47. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2018.08.002. URL: http://www.sciencedirect.com/science/article/pii/S0893608018302223.

[63] C. Gallicchio, A. Micheli, and L. Pedrelli. "Hierarchical Temporal Representation in Linear Reservoir Computing". In: *Neural Advances in Processing Nonlinear Dynamic Signals*. Ed. by A. Esposito, M. Faundez-Zanuy, F. C. Morabito, and E. Pasero. WIRN 2017, arXiv preprint arXiv:1705.05782. Cham: Springer International Publishing, 2019, pp. 119–129. ISBN: 978-3-319-95098-3. DOI: 10.1007/978-3-319-95098-3_11.

[64] C. Gallicchio, A. Micheli, L. Pedrelli, L. Fortunati, F. Vozzi, and O. Parodi. "A Reservoir Computing Approach for Balance Assessment". In: *Advanced Analysis and Learning on Temporal Data: First ECML PKDD Workshop, AALTD 2015, Porto, Portugal, September 11, 2015, Revised Selected Papers*. Ed. by A. Douzal-Chouakria, J.A. Vilar, and P.-F. Marteau. Vol. 9785. Lecture Notes in Computer Science. Springer International Publishing, 2016, pp. 65–77. ISBN: 978-3-319-44412-3. DOI: 10.1007/978-3-319-44412-3_5. URL: http://dx.doi.org/10.1007/978-3-319-44412-3_5.

[65] C. Gallicchio, A. Micheli, L. Pedrelli, F. Vozzi, and O. Parodi. "Preliminary Experimental Analysis of Reservoir Computing Approach for Balance Assessment". In: *Proceedings of the 1st International Workshop on Advanced Analytics and Learning on Temporal Data (AALTD)*. (Porto, Portugal). Ed. by A. Douzal-Chouakria et al. CEUR Workshop Proceedings 1425. 2015, pp. 57–62.

[66] C. Gallicchio, A. Micheli, and L. Silvestri. "Local Lyapunov Exponents of Deep Echo State Networks". In: *Neurocomputing* 298 (2018), pp. 34–45.

[67] C. Gallicchio, A. Micheli, and L. Silvestri. "Local Lyapunov Exponents of Deep RNN". In: *Proceedings of the 25th European Symposium on Artificial Neural Networks (ESANN)*. i6doc.com. 2017, pp. 559–564.

[68] C. Gallicchio, A. Micheli, and P. Tiňo. "Randomized Recurrent Neural Networks". In: *Proceedings of the 26th European Symposium on Artificial Neural Networks (ESANN)*. i6doc.com. 2018, pp. 415–424.

[69] J. Gil-Gómez, R. Lloréns, M. Alcañiz, and C. Colomer. "Effectiveness of a Wii balance board-based system (eBaViR) for balance rehabilitation: a pilot randomized clinical trial in patients with acquired brain injury." In: *Journal of neuroengineering and rehabilitation* 8.1 (2011), p. 30. ISSN: 1743-0003. DOI: 10.1186/1743-0003-8-30. URL: http://jneuroengrehab.biomedcentral.com/articles/10.1186/1743-0003-8-30.

[70] X. Glorot and Y. Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pp. 249–256.

[71] X. Glorot, A. Bordes, and Y. Bengio. "Deep Sparse Rectifier Neural Networks." In: *Aistats*. Vol. 15. 106. 2011, p. 275.

[72] I. Goodfellow, Y. Bengio, and A. Courville. "Deep Learning". Book in preparation for MIT Press. 2016. URL: http://www.deeplearningbook.org.

[73] A. Graves, A. Mohamed, and G. Hinton. "Speech recognition with deep recurrent neural networks". In: *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE. 2013, pp. 6645–6649.

[74] A. Graves and J. Schmidhuber. "Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks". In: *Advances in Neural Information Processing Systems 21*. Ed. by D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou. Curran Associates, Inc., 2009, pp. 545–552. URL: http://papers.nips.cc/paper/3449-offline-handwriting-recognition-with-multidimensional-recurrent-neural-networks.pdf.

[75] K. Gregor, I. Danihelka, A. Graves, D. Jimenez Rezende, and D. Wierstra. "Draw: A recurrent neural network for image generation". In: *arXiv preprint arXiv:1502.04623* (2015).

[76] M. A. Hajnal and A. Lőrincz. "Critical echo state networks". In: *International Conference on Artificial Neural Networks*. Springer. 2006, pp. 658–667.

[77] B. Hammer and P. Tiňo. "Recurrent neural networks with small weights implement definite memory machines". In: *Neural Computation* 15.8 (2003), pp. 1897–1929.

[78] C. Hartland and N. Bredeche. "Using echo state networks for robot navigation behavior acquisition". In: *ROBIO 07*. 2007, pp. 201–206.

[79] S. Haykin. *Neural networks and learning machines*. Vol. 3. Pearson Education Upper Saddle River, 2009.

[80] K. He, X. Zhang, S. Ren, and J. Sun. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[81] S. Hellbach, S. Strauss, J. P. Eggert, E. Körner, and H. Gross. “Echo State Networks for Online Prediction of Movement Data–Comparing Investigations”. In: *International Conference on Artificial Neural Networks*. Springer. 2008, pp. 710–719.

[82] M. Hermans and B. Schrauwen. “Training and analysing deep recurrent neural networks”. In: *Advances in neural information processing systems*. 2013, pp. 190–198.

[83] S. El Hihi and Y. Bengio. “Hierarchical Recurrent Neural Networks for Long-Term Dependencies.” In: *Advances in neural information processing systems*. 1995, pp. 493–499.

[84] G. E. Hinton. “Training products of experts by minimizing contrastive divergence”. In: *Neural computation* 14.8 (2002), pp. 1771–1800.

[85] G. E Hinton, S. Osindero, and Y. Teh. “A fast learning algorithm for deep belief nets”. In: *Neural computation* 18.7 (2006), pp. 1527–1554.

[86] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. “Improving neural networks by preventing co-adaptation of feature detectors”. In: *arXiv preprint arXiv:1207.0580* (2012).

[87] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. “Gradient flow in recurrent nets: the difficulty of learning long-term dependencies”. In: *A Field Guide to Dynamical Recurrent Neural Networks*. Ed. by S. C. Kremer and J. F. Kolen. IEEE Press, 2001.

[88] S. Hochreiter and J. Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[89] G. Holzmann and H. Hauser. “Echo state networks with filter neurons and a delay&sum readout”. In: *Neural Networks* 23.2 (2010), pp. 244–256.

[90] J. E. Hopcroft. *Introduction to Automata Theory, Languages and Computation: For VTU, 3/e*. Pearson Education India, 2013.

[91] K. Hornik. “Approximation capabilities of multilayer feedforward networks”. In: *Neural networks* 4.2 (1991), pp. 251–257.

[92] G. Huang, Q. Zhu, and C. Siew. “Extreme learning machine: theory and applications”. In: *Neurocomputing* 70.1 (2006), pp. 489–501.

[93] M. Isenkul, B. Sakar, and O. Kursun. “Improved spiral test using digitized graphics tablet for monitoring Parkinson’s disease”. In: *ICEHTM*. 2014, pp. 171–175.

[94] J. M. Jacob. *Advanced AC circuits and electronics: principles & applications*. Cengage Learning, 2004.

[95] H. Jaeger. “Discovering multiscale dynamical features with hierarchical Echo State Networks”. In: (2007). ISSN: 10. URL: http://jpubs.jacobs-university.de/handle/579/147.

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

[97] H. Jaeger. *Short term memory in echo state networks*. GMD-Forschungszentrum Informationstechnik, 2001.

[98] H. Jaeger. “The "echo state" approach to analysing and training recurrent neural networks”. In: *GMD - German National Research Institute for Computer Science, Tech. Rep.* (2001).

[99] H. Jaeger and H. Haas. “Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication”. In: *Science* 304.5667 (2004), pp. 78–80.

[100] H. Jaeger, M. Lukoševičius, D. Popovici, and U. Siewert. “Optimization and applications of echo state networks with leaky-integrator neurons”. In: *Neural Networks* 20.3 (2007), pp. 335–352.

[101] A. Jalalvand, G. Van Wallendael, and R. Van de Walle. “Real-time reservoir computing network-based systems for detection tasks on visual contents”. In: *Computational Intelligence, Communication Systems and Networks (CICSyN), 2015 7th International Conference on*. IEEE. 2015, pp. 146–151.

[102] J. Jankovic. “Parkinson’s disease: clinical features and diagnosis”. In: *Journal of Neurology, Neurosurgery & Psychiatry* 79.4 (2008), pp. 368–376.

[103] J. P. Jordanou, E. A. Antonelo, E. Camponogara, S. de Aguiar, and M. Aurelio. “Recurrent Neural Network based control of an Oil Well”. In: *Brazilian Symposium on Intelligent Automation, Porto Alegre 1-4 October 2017*. 2017.

[104] G. Juneja, J. J. Czyrny, and R. T. Linn. "Admission balance and outcomes of patients admitted for acute impatient rehabilitation". In: *American journal of physical medicine & rehabilitation* 77.5 (1998), pp. 388–393.

[105] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. "Large-scale video classification with convolutional neural networks". In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2014, pp. 1725–1732.

[106] D. Kinga and J. B. Adam. "A method for stochastic optimization". In: *International Conference on Learning Representations (ICLR)*. Vol. 5. 2015.

[107] K. C. Kiwiel. "Convergence and efficiency of subgradient methods for quasiconvex minimization". In: *Mathematical programming* 90.1 (2001), pp. 1–25.

[108] J. F. Kolen and S. C. Kremer. *A field guide to dynamical recurrent networks*. John Wiley & Sons, 2001.

[109] D. Koryakin, J. Lohmann, and M.V. Butz. "Balanced echo state networks". In: *Neural Networks* 36 (2012), pp. 35–45.

[110] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[111] R. Kumar and S. Vassilvitskii. "Generalized distances between rankings". In: *Proceedings of the 19th international conference on World wide web*. ACM. 2010, pp. 571–580.

[112] J. Kuwabara, K. Nakajima, R. Kang, D. T. Branson, E. Guglielmino, D. G. Caldwell, and R. Pfeifer. "Timing-based control via echo state network for soft robotic arm". In: *Neural Networks (IJCNN), The 2012 International Joint Conference on*. IEEE. 2012, pp. 1–8.

[113] S. E. Lacy, S. L. Smith, and M. A. Lones. "Using echo state networks for classification: A case study in Parkinson's disease diagnosis". In: *Artificial intelligence in medicine* 86 (2018), pp. 53–59.

[114] J. M. Leach, M. Mancini, R. J. Peterka, T. L. Hayes, and F. B. Horak. "Validating and calibrating the Nintendo Wii balance board to derive reliable center of pressure measures". In: *Sensors (Switzerland)* 14.10 (2014), pp. 18244–18267. ISSN: 14248220. DOI: 10.3390/s141018244.

[115] Y. LeCun and Y. Bengio. “Convolutional networks for images, speech, and time series”. In: *The handbook of brain theory and neural networks* 3361.10 (1995), p. 1995.

[116] Y. LeCun, Y. Bengio, and G. Hinton. “Deep learning”. In: *nature* 521.7553 (2015), p. 436.

[117] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[118] M. Leshno, V. Ya Lin, A. Pinkus, and S. Schocken. “Multilayer feedforward networks with a nonpolynomial activation function can approximate any function”. In: *Neural networks* 6.6 (1993), pp. 861–867.

[119] X. Lin, Z. Yang, and Y. Song. “Short-term stock price prediction based on echo state networks”. In: *Expert systems with applications* 36.3 (2009), pp. 7313–7317.

[120] M. Lukoševičius. “A practical guide to applying echo state networks”. In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 659–686.

[121] M. Lukoševičius and H. Jaeger. “Reservoir computing approaches to recurrent neural network training”. In: *Computer Science Review* 3.3 (2009), pp. 127–149.

[122] R. Lyon. “A computational model of filtering, detection, and compression in the cochlea”. In: *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP’82*. Vol. 7. IEEE. 1982, pp. 1282–1285.

[123] R. Legensteinand W. Maass. “Edge of chaos and prediction of computational performance for neural circuit models”. In: *Neural Networks* 20.3 (2007), pp. 323–334.

[124] W. Maass. “Bounds for the computational power and learning complexity of analog neural nets”. In: *SIAM Journal on Computing* 26.3 (1997), pp. 708–732.

[125] W. Maass, T. Natschläger, and H. Markram. “Real-time computing without stable states: A new framework for neural computation based on perturbations”. In: *Neural computation* 14.11 (2002), pp. 2531–2560.

[126] B. E. Maki, P. J. Holliday, and A. K. Topper. “A prospective study of postural balance and risk of falling in an ambulatory and independent elderly population”. In: *Journal of gerontology* 49.2 (1994), pp. M72–M84.

[127] Z.K. Malik, A. Hussain, and Q.J. Wu. "Multilayered echo state machine: a novel architecture and algorithm". In: *IEEE Transactions on cybernetics* 47.4 (2017), pp. 946–959.

[128] P. L. McDermott and C. K. Wikle. "An ensemble quadratic echo state network for non-linear spatio-temporal forecasting". In: *Stat* 6.1 (2017), pp. 315–330.

[129] S McNally, J Roche, and S Caton. "Predicting the price of Bitcoin using Machine Learning". In: *Parallel, Distributed and Network-based Processing (PDP), 2018 26th Euromicro International Conference on*. IEEE. 2018, pp. 339–343.

[130] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell. *Machine learning: An artificial intelligence approach*. Springer Science & Business Media, 2013.

[131] L. Mici, X. Hinaut, and S. Wermter. "Activity recognition with echo state networks using 3D body joints and objects category". In: *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*. 2016, pp. 465–470.

[132] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur. "Recurrent neural network based language model". In: *Eleventh Annual Conference of the International Speech Communication Association*. 2010.

[133] V. Mnih, K. Kavukcuoglu, D. Silver, A. A Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533.

[134] A. Mohamed, G. E Dahl, and G. Hinton. "Acoustic modeling using deep belief networks". In: *IEEE Transactions on Audio, Speech, and Language Processing* 20.1 (2012), pp. 14–22.

[135] A. Molund. "Deep Reservoir Computing Using Cellular Automata". MA thesis. NTNU, 2017.

[136] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio. "On the number of linear regions of deep neural networks". In: *Advances in neural information processing systems*. 2014, pp. 2924–2932.

[137] M.C. Mozer. "Induction of multiscale temporal structure". In: *Advances in neural information processing systems* (1993), pp. 275–275.

[138] S. Nichele and A. Molund. “Deep reservoir computing using cellular automata”. In: *arXiv preprint arXiv:1703.02806* (2017).

[139] S. Otte, M. V. Butz, D. Koryakin, F. Becker, M. Liwicki, and A. Zell. “Optimizing recurrent reservoirs with neuro-evolution”. In: *Neurocomputing* 192 (2016), pp. 128–138.

[140] F. Palumbo, C. Gallicchio, R. Pucci, and A. Micheli. “Human activity recognition using multisensor data fusion based on reservoir computing”. In: *Journal of Ambient Intelligence and Smart Environments* 8.2 (2016), pp. 87–107.

[141] F. Palumbo, D. La Rosa, E. Ferro, D. Bacciu, C. Gallicchio, A. Micheli, S. Chessa, F. Vozzi, and O. Parodi. “Reliability and human factors in ambient assisted living environments”. In: *Journal of Reliable Intelligent Environments* 3.3 (2017), pp. 139–157.

[142] Y. Pao and Y. Takefuji. “Functional-link net computing: theory, system architecture, and functionalities”. In: *Computer* 25.5 (1992), pp. 76–79.

[143] L. Pasa and A. Sperduti. “Pre-training of recurrent neural networks via linear autoencoders”. In: *Advances in Neural Information Processing Systems*. 2014, pp. 3572–3580.

[144] R. Pascanu, Ç. Gülçehre, K. Cho, and Y. Bengio. “How to construct deep recurrent neural networks”. In: *Proceedings of the Second International Conference on Learning Representations*. 2014.

[145] R. Pascanu, Ç. Gülçehre, K. Cho, and Y. Bengio. “How to construct deep recurrent neural networks”. In: *Proceedings of the Second International Conference on Learning Representations*. 2014.

[146] R. Pascanu, T. Mikolov, and Y. Bengio. “On the difficulty of training recurrent neural networks”. In: *International Conference on Machine Learning*. 2013, pp. 1310–1318.

[147] C. R. Pereira, D. R. Pereira, J. P. Papa, G. H. Rosa, and X. Yang. “Convolutional Neural Networks Applied for Parkinson’s Disease Identification”. In: *Machine Learning for Health Informatics*. Springer, 2016, pp. 377–390.

[148] P. G. Plöger, A. Arghir, T. Günther, and R. Hosseiny. “Echo state networks for mobile robot modeling and control”. In: *Robot Soccer World Cup*. Springer. 2003, pp. 157–168.

[149] C. Prahm, A. Schulz, B. Paaßen, O. Aszmann, B. Hammer, and G. Dorffner. "Echo State Networks as Novel Approach for Low-Cost Myoelectric Control". In: *Artificial Intelligence in Medicine*. Ed. by A. ten Teije, C. Popow, J. H. Holmes, and L. Sacchi. Cham: Springer International Publishing, 2017, pp. 338–342. ISBN: 978-3-319-59758-4.

[150] D. Prokhorov. "Echo state networks: appeal and challenges". In: *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*. Vol. 3. IEEE, pp. 1463–1466.

[151] J. Qiao, F. Li, H. Han, and W. Li. "Growing echo-state network with multiple subreservoirs". In: *IEEE transactions on neural networks and learning systems* 28.2 (2017), pp. 391–404.

[152] A. A. Qutubuddin, P.O. Pegg, D. X. Cifu, R. Brown, S. McNamee, and W. Carne. "Validating the Berg Balance Scale for patients with Parkinson's disease: a key to rehabilitation evaluation". In: *Archives of physical medicine and rehabilitation* 86.4 (2005), pp. 789–792.

[153] D. L. Riddle and P. W. Stratford. "Interpreting validity indexes for diagnostic tests: an illustration using the Berg balance test". In: *Physical therapy* 79.10 (1999), pp. 939–948.

[154] A. Rodan and P. Tiňo. "Minimum complexity echo state network". In: *IEEE transactions on neural networks* 22.1 (2011), pp. 131–144.

[155] A. Rodan and P. Tiňo. "Simple deterministically constructed cycle reservoirs with regular jumps". In: *Neural computation* 24.7 (2012), pp. 1822–1852.

[156] F. Rosenblatt. *Principles of neurodynamics. perceptrons and the theory of brain mechanisms*. Tech. rep. DTIC Document, 1961.

[157] F. Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[158] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning internal representations by error propagation*. Tech. rep. DTIC Document, 1985.

[159] H. Sak, A. W. Senior, and F. Beaufays. "Long short-term memory recurrent neural network architectures for large scale acoustic modeling." In: 2014.

[160] M. Salmen and P. G. Ploger. "Echo state networks used for motor control". In: *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE. 2005, pp. 1953–1958.

[161] R. Saunders-Pullman, C. Derby, K. Stanley, A. Floyd, S. Bressman, R. B. Lipton, A. Deligtisch, L. Severt, Q. Yu, M. Kurtis, and S. L. Pullman. "Validity of spiral analysis in early Parkinson's disease". In: *Movement disorders* 23.4 (2008), pp. 531–537.

[162] A. M. Saxe, J. L. McClelland, and S. Ganguli. "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks". In: *arXiv preprint arXiv:1312.6120* (2013).

[163] S. Scardapane and D. Wang. "Randomness in neural networks: an overview". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 7.2 (2017), e1200.

[164] N. Schaetti, M. Salomon, and R. Couturier. "Echo state networks-based reservoir computing for mnist handwritten digits recognition". In: *Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES), 2016 IEEE Intl Conference on*. IEEE. 2016, pp. 484–491.

[165] J. Schmidhuber. "Deep learning in neural networks: An overview". In: *Neural networks* 61 (2015), pp. 85–117.

[166] J. Schmidhuber. "Deep learning in neural networks: An overview". In: *Neural Networks* 61 (2015), pp. 85–117.

[167] J. Schmidhuber. "Learning complex, extended sequences using the principle of history compression". In: *Neural Computation* 4.2 (1992), pp. 234–242.

[168] J. Schmidhuber, D. Wierstra, M. Gagliolo, and F. Gomez. "Training recurrent networks by evolino". In: *Neural computation* 19.3 (2007), pp. 757–779.

[169] W. F. Schmidt, M. A Kraaijveld, and R. P. Duin. "Feedforward neural networks with random weights". In: *Pattern Recognition, 1992. Vol. II. Conference B: Pattern Recognition Methodology and Systems, Proceedings., 11th IAPR International Conference on*. IEEE. 1992, pp. 1–4.

[170] B. Schrauwen, J. Defour, D. Verstraeten, and J. Van Campenhout. "The introduction of time-scales in reservoir computing, applied to isolated digits recognition". In: *International Conference on Artificial Neural Networks*. Springer. 2007, pp. 471–479.

[171] B. Schrauwen, M. Wardermann, D. Verstraeten, J. J. Steil, and D. Stroobandt. "Improving reservoirs using intrinsic plasticity". In: *Neurocomputing* 71.7 (2008), pp. 1159–1171.

[172] C. H. Shih, C. T. Shih, and M. S. Chiang. "A new standing posture detector to enable people with multiple disabilities to control environmental stimulation by changing their standing posture through a commercial Wii Balance Board". In: *Research in Developmental Disabilities* 31.1 (2010), pp. 281–286. ISSN: 08914222. DOI: 10.1016/j.ridd.2009.09.013.

[173] A. Shumway-Cook, M. Baldwin, N. L. Polissar, and W. Gruber. "Predicting the probability for falls in community-dwelling older adults". In: *Physical therapy* 77.8 (1997), pp. 812–819.

[174] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (2016), pp. 484–489.

[175] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. "Mastering the game of Go without human knowledge". In: *Nature* 550.7676 (2017), p. 354.

[176] H. Simila, J. Mantyjarvi, J. Merilahti, M. Lindholm, and M. Ermes. "Accelerometry-based Berg Balance Scale score estimation". In: *Biomedical and Health Informatics, IEEE Journal of* 18.4 (2014), pp. 1114–1121.

[177] J. J. Steil. "Backpropagation-decorrelation: online recurrent learning with O (N) complexity". In: *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*. Vol. 2. IEEE. 2004, pp. 843–848.

[178] J. J. Steil. "Online reservoir adaptation by intrinsic plasticity for backpropagation–decorrelation and echo state learning". In: *Neural Networks* 20.3 (2007), pp. 353–364.

[179] J. J. Steil. "Online stability of backpropagation–decorrelation recurrent learning". In: *Neurocomputing* 69.7 (2006), pp. 642–650.

[180] W.S. Stornetta, T. Hogg, and B.A. Huberman. "A dynamical approach to temporal pattern processing". In: *Advances in neural information processing systems*. 1988, pp. 750–759.

[181] T. Strauss, W. Wustlich, and R. Labahn. “Design strategies for weight matrices of echo state networks”. In: *Neural computation* 24.12 (2012), pp. 3246–3276.

[182] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. “On the importance of initialization and momentum in deep learning”. In: *International conference on machine learning*. 2013, pp. 1139–1147.

[183] I. Sutskever, J. Martens, and G. Hinton. “Generating text with recurrent neural networks”. In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*. Omnipress. 2011, pp. 1017–1024.

[184] I. Sutskever, J. Martens, and G. E. Hinton. “Generating text with recurrent neural networks”. In: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 2011, pp. 1017–1024.

[185] I. Sutskever, O. Vinyals, and Q. V. Le. “Sequence to sequence learning with neural networks”. In: *Advances in neural information processing systems*. 2014, pp. 3104–3112.

[186] I. Sutskever, O. Vinyals, and Q. V. Le. “Sequence to Sequence Learning with Neural Networks”. In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger. Curran Associates, Inc., 2014, pp. 3104–3112. URL: http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf.

[187] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. “Going deeper with convolutions”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 1–9.

[188] M. E. Tinetti, S. K. Inouye, T. M. Gill, and J. T. Doucette. “Shared risk factors for falls, incontinence, and functional dependence: unifying the approach to geriatric syndromes”. In: *Jama* 273.17 (1995), pp. 1348–1353.

[189] P. Tiňo, M. Čerňanskỳ, and L. Beňušková. “Markovian architectural bias of recurrent neural networks”. In: *Neural Networks, IEEE Transactions on* 15.1 (2004), pp. 6–15.

[190] P. Tino and G. Dorffner. "Predicting the future of discrete sequences from fractal representations of the past". In: *Machine Learning* 45.2 (2001), pp. 187–217.

[191] P. Tiňo and B. Hammer. "Architectural Bias in Recurrent Neural Networks—Fractal Analysis". In: *International Conference on Artificial Neural Networks*. Springer. 2002, pp. 1359–1364.

[192] P. Tiňo, B. Hammer, and M. Bodén. "Markovian bias of neural-based architectures with feedback connections". In: *Perspectives of neural-symbolic integration*. Springer, 2007, pp. 95–133.

[193] F. Triefenbach, A. Jalalvand, K. Demuynck, and J. Martens. "Acoustic modeling with hierarchical reservoirs". In: *IEEE Transactions on Audio, Speech, and Language Processing* 21.11 (2013), pp. 2439–2450.

[194] F. Triefenbach, A. Jalalvand, B. Schrauwen, and J. Martens. "Phoneme recognition with large hierarchical reservoirs". In: *Advances in neural information processing systems*. 2010, pp. 2307–2315.

[195] J. Triesch. "A gradient rule for the plasticity of a neuron's intrinsic excitability". In: *International Conference on Artificial Neural Networks*. Springer. 2005, pp. 65–70.

[196] A. C. Tsoi. "Gradient based learning methods". In: *Adaptive processing of sequences and data structures*. Springer, 1998, pp. 27–62.

[197] C. H. Valencia, M. M. Vellasco, and K. T. Figueiredo. "Trajectory Tracking Control Using Echo State Networks for the CoroBot's Arm". In: *Robot Intelligence Technology and Applications 2*. Springer, 2014, pp. 433–447.

[198] V. N. Vapnik. "An overview of statistical learning theory". In: *IEEE transactions on neural networks* 10.5 (1999), pp. 988–999.

[199] V. N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998.

[200] D. Verstraeten, B. Schrauwen, M. d'Haene, and D. Stroobandt. "An experimental unification of reservoir computing methods". In: *Neural Networks* 20.3 (2007), pp. 391–403.

[201] D. Verstraeten, B. Schrauwen, D. Stroobandt, and J. Van Campenhout. "Isolated word recognition with the liquid state machine: a case study". In: *Information Processing Letters* 95.6 (2005), pp. 521–528.

[202] D. Wang and M. Li. “Stochastic configuration networks: Fundamentals and algorithms”. In: *IEEE transactions on cybernetics* 47.10 (2017), pp. 3466–3479.

[203] P. J. Werbos. “Generalization of backpropagation with application to a recurrent gas market model”. In: *Neural Networks* 1.4 (1988), pp. 339–356.

[204] O. L. White, D. D. Lee, and H. Sompolinsky. “Short-term memory in orthogonal neural networks”. In: *Physical review letters* 92.14 (2004), p. 148102.

[205] B. Widrow, A. Greenblatt, Y. Kim, and D. Park. “The No-Prop algorithm: A new learning algorithm for multilayer neural networks”. In: *Neural Networks* 37 (2013), pp. 182–188.

[206] D. Wierstra, F. J. Gomez, and J. Schmidhuber. “Modeling systems with internal state using evolino”. In: *Proceedings of the 7th annual conference on Genetic and evolutionary computation*. ACM. 2005, pp. 1795–1802.

[207] A. Woodward and T. Ikegami. “A reservoir computing approach to image classification using coupled echo state and back-propagation neural networks”. In: *Proc. of 26th Int. Conf. on Image and Vision Computing, Auckland, New Zealand, November*. 2011, pp. 543–458.

[208] J. Wu, Y. Wang, J. Huang, and H. Zhou. “Nonlinear Internal Model Control Using Echo State Network for Pneumatic Muscle System.” In: *JCP* 7.12 (2012), pp. 3060–3067.

[209] F. Wyffels, B. Schrauwen, D. Verstraeten, and D. Stroobandt. “Band-pass reservoir computing”. In: *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. IEEE. 2008, pp. 3204–3209.

[210] M. Xu, Y. Yang, M. Han, T. Qiu, and H. Lin. “Spatio-Temporal Interpolated Echo State Network for Meteorological Series Prediction”. In: *IEEE transactions on neural networks and learning systems* (2018).

[211] Y. Xue, L. Yang, and S. Haykin. “Decoupled echo state networks with lateral inhibition”. In: *Neural Networks* 20.3 (2007), pp. 365–376.

[212] Y. Yamashita and J. Tani. “Emergence of functional hierarchy in a multiple timescale neural network model: a humanoid robot experiment”. In: *PLoS computational biology* 4.11 (2008), e1000220.

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

[214] W. Young, S. Ferguson, S. Brault, and C. Craig. "Assessing and training standing balance in older adults: a novel approach using the Nintendo Wii Balance Board". In: *Gait & posture* 33.2 (2011), pp. 303–305.

[215] B. Zajzon, R. Duarte, and A. Morrison. "Transferring State Representations in Hierarchical Spiking Neural Networks". In: (2018).

[216] M. D. Zeiler and R. Fergus. "Visualizing and understanding convolutional networks". In: *European conference on computer vision*. Springer. 2014, pp. 818–833.

[217] P. Zham, D. K. Kumar, P. Dabnichki, S. Poosapadi Arjunan, and S. Raghav. "Distinguishing Different Stages of Parkinson's Disease Using Composite Index of Speed and Pen-Pressure of Sketching a Spiral". In: *Frontiers in Neurology* 8 (2017), p. 435. ISSN: 1664-2295. DOI: 10.3389/fneur.2017.00435. URL: https://www.frontiersin.org/article/10.3389/fneur.2017.00435.

[218] S. Zhang, A. E. Choromanska, and Y. LeCun. "Deep learning with elastic averaging SGD". In: *Advances in Neural Information Processing Systems*. 2015, pp. 685–693.

[219] S. Zhang, Y. Wu, T. Che, Z. Lin, R. Memisevic, R. R. Salakhutdinov, and Y. Bengio. "Architectural complexity measures of recurrent neural networks". In: *Advances in Neural Information Processing Systems*. 2016, pp. 1822–1830.