Neural Networks 108 (2018) 33–47

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Design of deep echo state networks

Claudio Gallicchio, Alessio Micheli *, Luca Pedrelli

Department of Computer Science, University of Pisa, Largo B. Pontecorvo 3, Pisa, Italy

HIGHLIGHTS

- We address the issue of architectural design for Deep Recurrent Neural Networks.
- We focus on the frequency analysis for layering design.
- We propose an efficient approach to choose the number of recurrent layers in DeepESN.
- We show the empirical advantage of very Deep (>30 recurrent layers) RNNs.

ARTICLE INFO

Article history:
Received 14 December 2017
Received in revised form 27 July 2018
Accepted 2 August 2018
Available online 8 August 2018

Keywords:
Reservoir computing
Echo state networks
Deep echo state networks
Deep recurrent neural networks
Architectural design of recurrent neural networks

ABSTRACT

In this paper, we provide a novel approach to the architectural design of deep Recurrent Neural Networks using signal frequency analysis. In particular, focusing on the Reservoir Computing framework and inspired by the principles related to the inherent effect of layering, we address a fundamental open issue in deep learning, namely the question of how to establish the number of layers in recurrent architectures in the form of deep echo state networks (DeepESNs). The proposed method is first analyzed and refined on a controlled scenario and then it is experimentally assessed on challenging real-world tasks. The achieved results also show the ability of properly designed DeepESNs to outperform RC approaches on a speech recognition task, and to compete with the state-of-the-art in time-series prediction on polyphonic music tasks.

1. Introduction

In the last years, the study of deep architectures aroused a great interest in the neural network research community. Based on a hierarchical organization of multiple layers, such networks proved effective in developing a compositional internal representation of the input information, allowing to address challenging real-world problems from several application fields featured by complex data (Goodfellow, Bengio, & Courville, 2016; Graves, Mohamed, & Hinton, 2013; Krizhevsky, Sutskever, & Hinton, 2012; Mohamed, Dahl, & Hinton, 2012; Sutskever, Vinyals, & Le, 2014). In particular, recent studies on deep recurrent neural networks (RNNs) opened a way to develop novel models able to learn hierarchical temporal representations from signals characterized by multiple time-scales dynamics (Angelov & Sperduti, 2016; Hermans & Schrauwen, 2013; Hihi & Bengio, 1995; Pascanu, Gülçehre, Cho, & Bengio, 2014; Schmidhuber, 2015).

In the field of randomized neural networks (Gallicchio, Martin-Guerrero, Micheli, & Soria-Olivas, 2017; Gallicchio, Micheli, & Tiňo, 2018), studies in the Reservoir Computing (RC) area (Lukoševičius & Jaeger, 2009; Verstraeten, Schrauwen, d'Haene, & Stroobandt, 2007) targeting ad-hoc modular organizations of Echo State Networks (ESNs) (Jaeger, 2001b; Jaeger & Haas, 2004), showed promising results on time-series tasks (see e.g. Jaeger, 2007; Malik, Hussain, & Wu, 2017; Triefenbach, Jalalvand, Demuynck, & Martens, 2013). Recently, the RC/ESN framework has been explicitly extended towards the deep learning framework, allowing to analyze the intrinsic aspects of layering in stacked RNN architectures, at the same time providing efficient solutions for building deep RNNs (Gallicchio, Micheli, & Pedrelli, 2017). Studies on the DeepESN model (Gallicchio, Micheli, & Pedrelli, 2017) shed light on the significance of stacking layers of recurrent units, pointing out the inherent characterization of the system dynamics developed at the different layers in deep RNNs. Empirical results in Gallicchio, Micheli, and Pedrelli (2017) and Gallicchio and Micheli (2016), as well as theoretical investigations in the field of dynamical systems (Gallicchio & Micheli, 2017b) and Lyapunov exponents (Gallicchio, Micheli, & Silvestri, 2017, 2018) highlighted the natural structured organization of the state dynamics developed by deep recurrent architectures even without training of the recurrent connections. Higher layers in the stack can progressively develop qualitatively

* Corresponding author.
E-mail address: micheli@di.unipi.it (A. Micheli).

https://doi.org/10.1016/j.neunet.2018.08.002
0893-6080/

different dynamics, and this natural differentiation results in a rich multiple time-scales representation of the temporal information. Moreover, the analysis in Gallicchio, Micheli, and Pedrelli (2019) noticeably showed that such hierarchical organization of the temporal features developed by the dynamics of stacked recurrent layers still holds even in case of linear activation functions. Further details on the analysis and advancements of DeepESN can be found in Gallicchio and Micheli (2017a).

Overall, the analysis conducted so far in the RC context 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 paper 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 Pascanu et al. (2014), 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 paper we propose an automatic method for the design of DeepESN, based on frequency analysis and aimed at appropriately exploiting the differentiation of temporal representation in deep recurrent architectures. The hypotheses (that delineate the scope of the work) 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. First, 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. Then, 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.

A further contribution of this work is to explicitly show, for the first time in the literature, the performance advantage on real-world tasks resulting from hierarchically structured recurrent state space organizations of deep RC models, through a comparative assessment with both fully trained neural network methodologies and RC-based approaches.

The paper is organized as follows. In Section 2 we first recall the basics of the standard (shallow) ESN approach (in Section 2.1) and then we introduce the DeepESN model (in Section 2.2). In Section 3 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. In Section 4 we evaluate our approach on music processing and speech processing tasks. We discuss the outcomes of our analysis in Section 5, and we present conclusions in Section 6. Further details on experimental results are provided in Appendix.

2. Deep reservoir computing

In this section, we introduce the deep RC framework. First, we briefly describe the standard shallow RC in Section 2.1. After that, we introduce deep RC architectures in Section 2.2.

2.1. Shallow echo state networks

The ESN (Jaeger & Haas, 2004) model is an efficient implementation of an RNN within the RC framework (Lukoševičius & Jaeger, 2009; Verstraeten et al., 2007). It is characterized by a recurrent layer called reservoir and by a linear output layer called readout. The reservoir implements a discrete-time dynamical system through untrained recurrent connections and it provides a suffix-based Markovian representation of the input history (Gallicchio & Micheli, 2011; Tiňo, Hammer, & Bodén, 2007). The readout is a linear layer that computes the output of the network by exploiting the temporal state representation developed by the reservoir part. In our work, we consider a variant of ESN called Leaky Integrator ESN (LI-ESN) (Jaeger, Lukoševičius, Popovici, & Siewert, 2007), in which the reservoir 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:

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

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, $\mathbf{W}_{in} \in \mathbb{R}^{N_R \times N_U}$ 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. The reservoir parameters are initialized in order to satisfy the Echo State Property (ESP) (Gallicchio & Micheli, 2011; Jaeger & Haas, 2004; Yildiz, Jaeger, & Kiebel, 2012) and after that they are left untrained. Accordingly, the values in matrix $\hat{\mathbf{W}}$ are randomly selected from a uniform distribution, e.g. in $[-1, 1]$, and then re-scaled in order to control spectral-related properties that influence the stability of network dynamics. Commonly, following the necessary condition for the ESP, and denoting by $\rho(\cdot)$ the spectral radius operator (i.e. the largest absolute eigenvalue of its matrix argument), the weight values in $\hat{\mathbf{W}}$ are re-scaled to meet the following condition:

$$\rho\left((1-a)\mathbf{I} + a\hat{\mathbf{W}}\right) < 1. \tag{2}$$

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

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

$$\mathbf{y}(t) = \mathbf{W}_{out}\mathbf{x}(t). \tag{3}$$

The readout layer is the only part of the network that is trained, typically using pseudo-inversion or ridge regression methods (Lukoševičius & Jaeger, 2009) computed through singular value decomposition (SVD). In the following, we refer to the LI-ESN model described so far as *shallowESN*.

In the context of RC, a well known approach for unsupervised adaptation of the reservoir is given by Intrinsic Plasticity (IP) (Lukoševičius & Jaeger, 2009; Schrauwen, Wardermann, Verstraeten, Steil, & Stroobandt, 2008; Steil, 2007; Triesch, 2005), which 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 of reservoir units to fit a desired distribution, which in case of *tanh* is of Gaussian type. In our context, the application of the reservoir activation function for a generic reservoir unit can be expressed as $\tilde{x} = tanh(g\, x_{net} + b)$, where x_{net}, $\tilde{x}$, g and b are respectively the (net) input, the output, the gain and the bias of the activation function. The IP training procedure is performed on each reservoir unit on the basis of the IP rule (Schrauwen et al., 2008) defined as:

$$\begin{aligned} \Delta b &= -\eta_{IP}(-(\mu_{IP}/\sigma_{IP}^2) + (\tilde{x}/\sigma_{IP}^2)(2\sigma_{IP}^2 + 1 - \tilde{x}^2 + \mu_{IP}\tilde{x})), \\ \Delta g &= \eta_{IP}/g + \Delta b\, x_{net}, \end{aligned} \tag{4}$$

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, Δg and Δb respectively denote the update values for gain and bias of the IP iterative learning algorithm.

2.2. Deep echo state networks

The DeepESN model, recently introduced in Gallicchio, Micheli, and Pedrelli (2017), allowed to frame the ESN approach in the context of deep learning. The architecture of a DeepESN is characterized by a stacked hierarchy of reservoirs, as shown in Fig. 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.

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

$$\begin{aligned} \mathbf{x}^{(1)}(t) = (1 - a^{(1)})\mathbf{x}^{(1)}(t-1) + a^{(1)}\mathbf{tanh}(\mathbf{W}_{in}\mathbf{u}(t) \\ + \hat{\mathbf{W}}^{(1)}\mathbf{x}^{(1)}(t-1)), \end{aligned} \tag{5}$$

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

$$\begin{aligned} \mathbf{x}^{(l)}(t) = (1 - a^{(l)})\mathbf{x}^{(l)}(t-1) + a^{(l)}\mathbf{tanh}(\mathbf{W}^{(l)}\mathbf{x}^{(l-1)}(t) \\ + \hat{\mathbf{W}}^{(l)}\mathbf{x}^{(l)}(t-1)), \end{aligned} \tag{6}$$

where $\mathbf{W}_{in} \in \mathbb{R}^{N_R \times N_U}$ 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 **tanh** represents the element-wise application of the hyperbolic tangent.

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 Gallicchio and Micheli (2017b). 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 deep reservoirs (Gallicchio & Micheli, 2017b), described by the following equation:

$$\max_{1 \le l \le N_L} \rho\left((1 - a^{(l)})\mathbf{I} + a^{(l)}\hat{\mathbf{W}}^{(l)}\right) = \max_{1 \le l \le N_L} \rho^{(l)} < 1, \tag{7}$$

in which we used the notation $\rho^{(l)}$ to denote the effective spectral radius of the reservoir system in the lth layer. As regards input and inter-layer matrices, the values in $\mathbf{W}_{in}$ and $\{\mathbf{W}^{(l)}\}_{l=2}^{N_L}$ are randomly initialized from a uniform distribution and then re-scaled in the $[-scale_{in}, scale_{in}]$ range.

Note that, 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 this paper, 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.

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 (Hermans & Schrauwen, 2013; Pascanu et al., 2014), in this paper we focus on the case represented in Fig. 1, in which the states of all the reservoir layers are used to feed the readout. Specifically, considering the global state of the DeepESN as the composition of the states in the different layers, i.e. $\mathbf{x}(t) = (\mathbf{x}^{(1)}(t), \ldots, \mathbf{x}^{(N_L)}(t)) \in \mathbb{R}^{N_L N_R}$, the output of the network at time t, i.e. $\mathbf{y}(t) \in \mathbb{R}^{N_Y}$, is computed as follows:

$$\mathbf{y}(t) = \mathbf{W}_{out}\mathbf{x}(t), \tag{8}$$

where $\mathbf{W}_{out} \in \mathbb{R}^{N_Y \times N_L N_R}$ denotes the matrix of the output weights. Note that in the case of DeepESN, the readout formulation given in Eq. (8) expresses a linear combination between the global reservoir state of the network $\mathbf{x}(t)$ and the readout weight matrix $\mathbf{W}_{out}$, 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

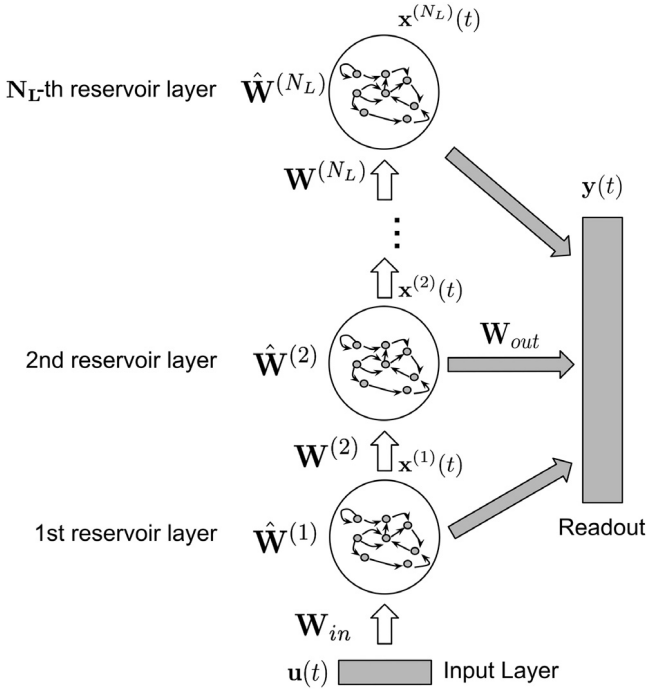

Fig. 1. Architecture of a DeepESN.

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. As regards the training algorithms, the output layer in DeepESNs is trained as in the standard RC paradigm (Lukoševičius & Jaeger, 2009), i.e. using pseudo-inversion or ridge regression implemented through SVD.

Although the DeepESN model has been introduced in the neural networks' literature only recently (Gallicchio & Micheli, 2016; Gallicchio, Micheli, & Pedrelli, 2017), the outcomes of its study already allowed to clearly highlight a number of major advantages that are *inherently* brought by a layered construction of RNN architectures (Gallicchio & Micheli, 2018), i.e. even prior to training of the recurrent connections. These advantages, which contribute to delineate the characterizations of the DeepESN approach, are briefly summarized in the following.

- *Multiple temporal representations.* 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 Gallicchio, Micheli, and Pedrelli (2017) and Gallicchio and Micheli (2016) 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 Gallicchio, Micheli, and Pedrelli (2017). In this regard, an interesting observation from the work in Gallicchio, Micheli, and Pedrelli (2017) 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 the work in Gallicchio, Micheli, and Pedrelli (2017) 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.
  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 (Xue, Yang, & Haykin, 2007), an idea that has been pursued also outside of the RC context e.g. in Yamashita and Tani (2008). A recent development, described in Qiao, Li, Han, and Li (2017), 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, already studied in Gallicchio, Micheli, and Pedrelli (2017), 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 multiplicity of temporal representations developed by the global internal state of a DeepESN has been also analyzed in terms of the frequency spectrum of state components in case of linear activation functions. In this case, results in Gallicchio et al. (2019) pointed out the natural propensity of DeepESN states to produce a *multiple frequency representation* of the input information, distributed through layers, whereas (in the conditions analyzed in Gallicchio et al., 2019) higher layers tended to focus on lower frequencies.
- *Richness of reservoir states.* 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 Gallicchio, Micheli, and Pedrelli (2017), by measuring the averaged entropy of DeepESN states, using the IP rule for unsupervised adaptation and in comparison to the shallow case. On the theoretical side, studies in the field of dynamical system theory (Gallicchio & Micheli, 2017b) 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 Gallicchio and Micheli (2017b) (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 Gallicchio, Micheli, and Silvestri (2017, 2018), pointed out that layered RNN architectures, compared to shallow counterparts in condition of equal number of recurrent units, show a dynamical behavior that is naturally pushed closer to the *edge of chaos*. 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 (Bertschinger & Natschläger, 2004; Maass, 2007).
- *Memory Capacity*. The hierarchical reservoir organization in DeepESNs has also a natural positive effect on the short-term memory abilities of the network, which in the context of RC is commonly evaluated by means of the Memory Capacity (MC) task defined in Jaeger (2001a). Experimental evidences reported in Gallicchio, Micheli, and Pedrelli (2017) showed that DeepESNs are able to considerably and consistently outperform corresponding shallow ESN settings (with the same total number of recurrent units) in terms of MC. A further performance gain on the MC task was found through the combination of DeepESN with IP adaptation, providing a concrete example of a case in which the introduction of depth in the RNN design is able to naturally enhance the advantages brought by IP in tasks (such as MC) on which this adaptation technique is already effective by itself (Schrauwen et al., 2008). Moreover, recent results given in Gallicchio (2018) indicated that the MC improvement that can be achieved by hierarchical reservoir models can be observed not only at a global scale, i.e. considering the entire network's global dynamics, but also at the level of individual layers' dynamics at increasing height in the architecture.
- *Efficiency.* From an architectural view-point, introducing a layered reservoir construction into the architectural RNN/ESN design has also the effect of reducing the number of non-zero recurrent connections. This contributes to a number of inherent implications on the characterization of the developed system dynamics, as discussed in the previous points. Besides, layering turns out to be a striking advantageous setting also under the perspective of computational efficiency (Gallicchio & Micheli, 2018), which in an RC context sums to the well-known efficiency of training algorithms. In particular, under the condition of a total number of recurrent units given by $N_L\, N_R$, while the cost of each

state update for shallow ESNs (Eq. (1)) amounts to $\mathcal{O}(N_L^2 N_R^2)$, in the case of DeepESNs (Eqs. (5) and (6)) it is reduced, by the sparsity of the connectivity given by the layering constraints, to $\mathcal{O}(N_L N_R^2)$. This straightforwardly implies a saving in the time complexity required by the reservoir operation that scales with the number of layers in which the same number of units is organized.

Finally, it is worth noticing that the use of IP in combination with deep reservoir organizations generally resulted effective in improving DeepESN performance under several aspects, as briefly recalled in the above points. In light of this observation, in this paper we use DeepESNs in synergy with the IP adaptation technique.

3. Spectral analysis and depth of DeepESN

In the following sections, we introduce our approach based on the spectral analysis for automatically determining the depth of DeepESN architectures. First, in Section 3.1 we define the iterative algorithm, then in Section 3.2 we discuss its advantages with respect to a standard cross-validation approach in terms of computational costs. In Section 3.3 we assess our method on a controlled scenario. Finally, in Section 3.4 we experimentally analyze the role of the depth in deep RNNs using spectral analysis.

3.1. Method

In this section, we 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. Recent research findings (Gallicchio, Micheli, & Pedrelli, 2017; Gallicchio et al., 2019) (briefly discussed in Section 2.2) showed that a stack of recurrent layers can develop a multiple time-scales differentiation among the layers even considering the same value of the leaky rate for each recurrent layer and unit. 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 (Holzmann & Hauser, 2010; Jaeger et al., 2007; Lukoševičius, 2012; Wyffels, Schrauwen, Verstraeten, & Stroobandt, 2008). 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 Eqs. (10) and (11)) 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 3.4). Therefore, we define a *stop condition* of the iterative algorithm to detect when the shift of spectral centroid converges:

$$|\mu^{(l)} - \mu^{(l-1)}| \leq \sigma^{(l-1)}\eta \tag{9}$$

where $0 < \eta < 1$ (see Section 3.4 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 Eq. (9), 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 (Eq. (10)) and the spectral spread (Eq. (11)) are defined as follows:

$$\mu^{(l)} = \left(\sum_{j=1}^{k} p_j^{(l)} f_j^{(l)}\right) / \sum_{j=1}^{k} p_j^{(l)} \tag{10}$$

$$\sigma^{(l)} = \sqrt{\left(\sum_{j=1}^{k} p_j^{(l)} (f_j^{(l)} - \mu^{(l)})^2\right) / \sum_{j=1}^{k} p_j^{(l)}}, \tag{11}$$

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 detailed in Algorithm 1.

Algorithm 1 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: $\mathbf{p}^{(l)} \leftarrow$ `mean(comps_g)` ▷ average on guesses (rows)
12: $\mathbf{f}^{(l)} \leftarrow$ [1 : $\lfloor$`timesteps/2`$\rfloor$`]/timesteps` ▷ normalized frequency (cyc/s)
13: **return** $\mathbf{p}^{(l)}, \mathbf{f}^{(l)}$

Algorithm 1 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 2 selects a number of layers for the network's architecture. The function *computeState*( ) called inside Algorithm 2, is composed by two steps, first, the IP Adaptation is performed (see Eq. (4)) 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 2 (i.e., Eq. (9)) is satisfied or the max number of layers (i.e., M_L) is reached.

Algorithm 2 Design of DeepESN

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

Algorithm 3 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

Algorithm 4 Cross-Validation with the Design Algorithm

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

3.2. 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 (Angelov & Sperduti, 2016). The aim of the analysis provided in this sub-section is to quantify the advantage, in terms of 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 3.

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

$$\mathcal{C}_{deep}(M_L) = \sum_{N_L=1}^{M_L} \mathcal{C}_{tr}(N_L), \tag{12}$$

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

Within the context of the deep RC paradigm, considered in this paper, the readout layer is the only part of the network that is trained. Accordingly, for DeepESNs the training cost in Algorithm 3, i.e. the $\mathcal{C}_{tr}$ term in Eq. (12), 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:

$$\mathcal{C}_{tr\_rc}(N_L) = \mathcal{O}((N_R N_L)^2 N_T). \tag{13}$$

Therefore, using the right-hand side of Eq. (13) as training cost in Eq. (12), 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 3 is given by:

$$\mathcal{C}_{basic}(M_L) = \sum_{N_L=1}^{M_L} \mathcal{O}((N_R N_L)^2 N_T) = \mathcal{O}(N_R^2 M_L^3 N_T). \tag{14}$$

Compared to the basic procedure explained above, the design methodology proposed in this paper 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 2. The resulting selection process is illustrated in Algorithm 4.

We can note that the cost of Algorithm 2 is dominated by the cost of performing the FFT, given by:

$$\mathcal{C}_{fft}(M_L) = \mathcal{O}(N_R M_L N_V log(N_V)), \tag{15}$$

where N_V is the size of the validation set. Overall, the cost entailed by the proposed Algorithm 4 is determined by the sum between $\mathcal{C}_{fft}(M_L)$ in Eq. (15), which is linear in the total number of recurrent units $N_R M_L$, and the cost $\mathcal{C}_{tr\_rc}(M_L)$ in Eq. (13), 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 4 (for each hyper-parameterization θ and fold `f`) can be expressed as follows:

$$\mathcal{C}_{design}(M_L) = \mathcal{O}(N_R^2 M_L^2 N_T). \tag{16}$$

Overall, comparing $\mathcal{C}_{design}$ and $\mathcal{C}_{basic}$ 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.

3.3. 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]

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

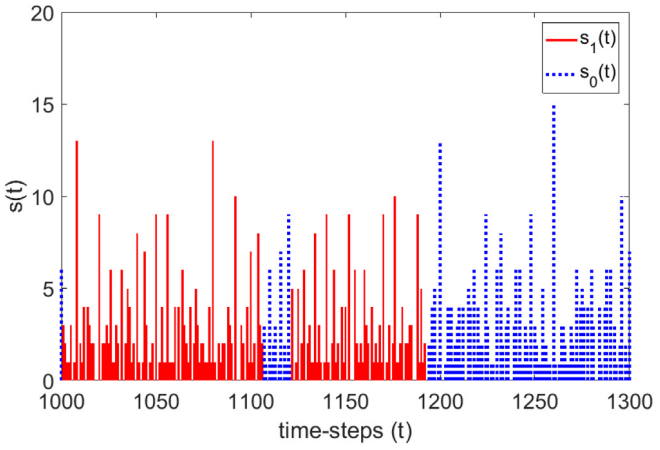

Fig. 2. A 300 time-step long excerpt of the sequence **s** for the FBC task.

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 **s** is defined as follows:

$$\mathbf{s} = [\mathbf{s}_0(0), \mathbf{s}_{j_1}(1), \ldots, \mathbf{s}_{j_n}(n)] \tag{17}$$

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 a probability of 0.01, in formulas:

$$j_{t+1} = \begin{cases} j_t + 1 \ mod \ 2 & \text{with probability } 0.01 \\ j_t & \text{otherwise.} \end{cases} \tag{18}$$

Fig. 2 shows an excerpt of the sequence **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 Eq. (18)) to have a switch between subsequences $\mathbf{s}_1$ and $\mathbf{s}_0$ is low, for instance in Fig. 2, this happens only in time-steps, 1107, 1122 and 1194. Therefore, sequence **s** tends to have long ranges with elements of the same subsequence $\mathbf{s}_i$. In particular, in the generated sequence **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.

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 $\mathbf{y}_{target}(t)$ target, contained in the labeled dataset $\{\mathbf{y}_{target}(t), \mathbf{s}(t)\}_{t=1}^{6000}$, is defined as follows:

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

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 **s**. We applied the design algorithm proposed in Section 3.1 to determine the number of DeepESN recurrent layers, considering the network's hyper-parameterizations as specified by the ranges reported in Table 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 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, $a^{(l)}$ and $\rho^{(l)}$ 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 2 with $\eta = 0.01$ The choice of such η value is motivated and discussed in Section 3.4.

Table 1
Range of DeepESN and shallowESN hyper-parameters values for model selection in the considered tasks.

Hyper-parameter	
Readout regularization λ_r	$0, 10^{-15}, 10^{-14}, \ldots, 10^{0}$
Input and inter-layer scaling $scale_{in}$	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 2
Training (TR), validation (VL) and test (TS) classification accuracy (ACC) obtained by DeepESN and shallowESN on the FBC task.

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 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.

The hyper-parameters that obtained the best performance on the validation set are $N_L = 7$ (by the design algorithm), $N_R = 100$, $\rho = 0.9$, $scale_{in} = 0.1$, $a = 0.1$, $\sigma_{IP} = 0.01$ and $\lambda_r = 10^{-10}$ for DeepESN and $N_L = 1$, $N_R = 300$, $\rho = 0.9$, $scale_{in} = 1$, $a = 0.1$, $\sigma_{IP} = 0.01$ and $\lambda_r = 10^{-7}$ for shallow ESN. Results in Table 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. Fig. 3 shows the classification errors (= 100 −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.

In Fig. 3, the marker 'X' represents the number of layers selected by the design algorithm (7 in our case-study). Noteworthy,

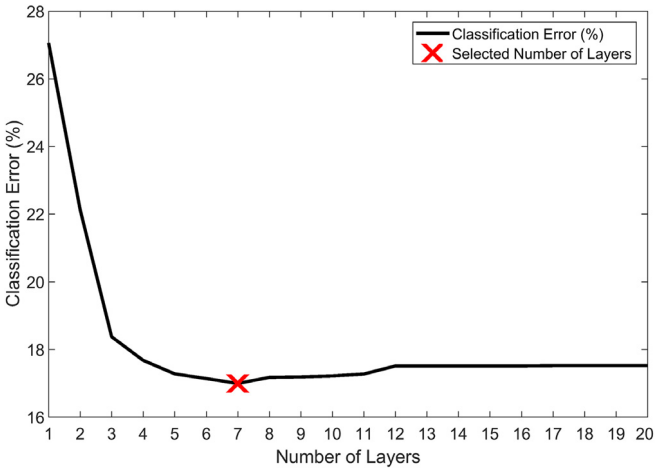

Fig. 3. Classification error obtained on the validation set of the FBC task by DeepESN architectures with a number of recurrent layers up to 20. Results are obtained through model selection individually performed for each number of layers. The marker 'X' indicates the number of recurrent layers selected by the design algorithm.

the design method added the optimal number of layers in the hierarchy, beside 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. Fig. 4(a), (b), (c) and (d) 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 $(1/3, 1/4, 1/5, \ldots, 1/31)$ 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 Fig. 4(a), (b), (c) and (d). In Fig. 4(a), 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(b), (c) and (d) 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 Eq. (9)) 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(b), (c) and (d), shifts progressively to the left in the higher layers towards low-frequency components. At layer 7 the condition in Eq. (9) 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 Fig. 4, 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.

3.4. Experimental analysis of depth in the controlled scenario

In this section, we empirically evaluate how the spectral centroid, computed by Algorithm 1 over the layers, varies with the depth of the stacked recurrent architecture. This allows us to assess the considered η value and to evaluate the effectiveness of the Algorithm 2 and the stop condition defined in Eq. (9).

A recurrent layer can be studied as a (low/high/band pass) filter (Holzmann & Hauser, 2010; Jaeger et al., 2007; Lukoševičius, 2012; Wyffels et al., 2008). 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 (Jacob, 2004), 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 Eq. (9) 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 behavior, 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 Sections 4.1 and 4.2), in Appendix.

Fig. 5 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.

As expected, the layers progressively apply a filter to the signal. In particular, in the case represented in Fig. 5, 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 (on the considered real-world tasks both the cases of low pass filters and high pass filters

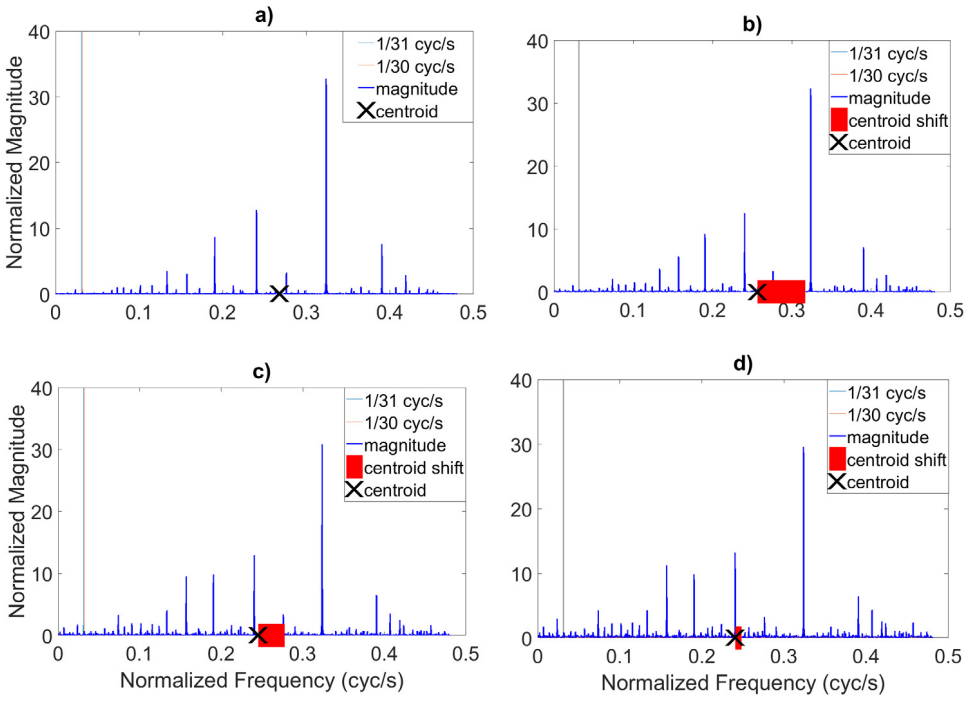

Fig. 4. Frequency components computed over time on reservoir states, encoding the sequence defined in Eq. (17), in progressively higher layers of DeepESN. (a) layer 1, (b) layer 3, (c) layer 5, (d) layer 7. The red range represents the shift of spectral centroid between current and previous layer, multiplied by a factor of 10. Normalized frequency is expressed in cycles per second (cyc/s). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

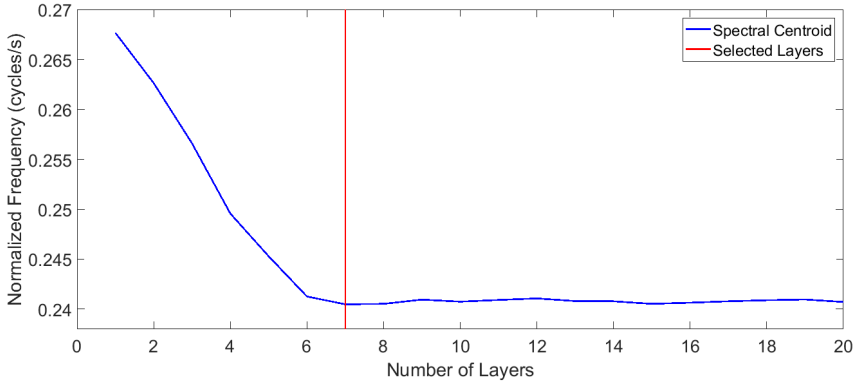

Fig. 5. Spectral centroid computed on the state of DeepESN layers optimized on FBC task. The red vertical line indicates the number of layers selected by the design algorithm. Threshold η value is set to 0.01. Normalized frequency is expressed in cycles per second (cyc/s).

are found — see Appendix). Moreover, as we can see in Fig. 5 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 7th layer, i.e. in correspondence to the optimal number of layers found for the task (see Fig. 3). Furthermore, from Fig. 5, 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 the convergence is smoother — see Appendix). 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 considered in this paper, as reported more in detail in Appendix.

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 paper, 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 Fig. 5) that can be helpful in assessing the useful (possibly different) η choice for the task at hand.

4. Experimental assessment on real-world tasks

In this section we present the experimental assessment on real-world tasks of the DeepESN approach (Section 2.2) constructed according to the design method proposed in Section 3. Specifically, results achieved on tasks in the areas of music processing and speech processing are respectively presented in Sections 4.1 and 4.2.

Table 3
The main characteristics of the preprocessed piano-rolls samples in training, validation and test set, as defined in Boulanger-Lewandowski et al. (2012).

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

4.1. Polyphonic music tasks

In this section, we evaluate our model on polyphonic music tasks defined in Boulanger-Lewandowski, Bengio, and Vincent (2012). In particular, we consider two datasets, namely Piano-midi.de[2] and MuseData.[3] These datasets are characterized by high dimensionality and complex temporal dependencies involved at different time-scales, forming interesting benchmarks for RNNs (Bengio, Boulanger-Lewandowski, & Pascanu, 2013). 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[4] of the authors of Boulanger-Lewandowski et al. (2012). Table 3 shows the main characteristics of Piano-midi.de and MuseData preprocessed piano-rolls as provided in Boulanger-Lewandowski et al. (2012).

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 Bay, Ehmann, and Downie (2009) and adopted in polyphonic music tasks in Boulanger-Lewandowski et al. (2012), computed as follows:

$$\text{FL-ACC} = \frac{\sum_{t=1}^{T} TP(t)}{\sum_{t=1}^{T} TP(t) + \sum_{t=1}^{T} FP(t) + \sum_{t=1}^{T} FN(t)}, \tag{19}$$

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 2 considering a number of maximum recurrent layers $M_L = 50$. All other aspects of experimental setup were as described in Section 3.3, and the remaining hyper-parameters were chosen (for model selection purposes) from the same ranges as shown in Table 1. To assess the effectiveness of the proposed methodology, we compared the performance achieved by DeepESNs built using the method proposed in Section 3, with the one obtained by shallowESNs, considering the same ranges for hyper-parameters values shown in Table 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 (Boulanger-Lewandowski et al., 2012), 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 (Boulanger-Lewandowski et al., 2012). 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 Pasa and Sperduti (2014), in which however the pre-processing of piano-rolls is different[5] and, as this affects the performance, the corresponding results are thereby difficultly comparable.

Table 4 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 $N_L = 35$, $N_R = 200$, $\rho = 0.1$, $scale_{in} = 0.1$, $a = 0.7$, $\sigma_{IP} = 0.1$ and $\lambda_r = 10^{-1}$ for DeepESN, and $N_L = 1$, $N_R = 5000$, $\rho = 0.5$, $scale_{in} = 0.01$, $a = 0.1$, $\sigma_{IP} = 0.1$ and $\lambda_r = 10^{-2}$ for shallow ESN. While for what regards MuseData task, the hyper-parameters that obtained the best performance on the validation set are $N_L = 36$, $N_R = 200$, $\rho = 0.1$, $scale_{in} = 0.1$, $a = 0.7$, $\sigma_{IP} = 0.1$ and $\lambda_r = 10^{-2}$ for DeepESN, and $N_L = 1$, $N_R = 6000$, $\rho = 0.3$, $scale_{in} = 0.01$, $a = 0.3$, $\sigma_{IP} = 0.1$ and $\lambda_r = 10^{-2}$ for shallow ESN. 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 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 recurrent 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 shallow ESN), the networks were selected on the basis of the performance achieved

[2] Classical piano MIDI archive www.piano-midi.de.

[3] Library of orchestral and piano classical music from CCARH www.musedata.org.

[4] www-etud.iro.umontreal.ca/boulanni/icml2012.

[5] Table 3 shows the characteristics of preprocessed piano-rolls provided by the authors of paper (Boulanger-Lewandowski et al., 2012) in which are defined the polyphonic music tasks. Note that the characteristics of preprocessed piano-rolls in Table 1 of paper (Pasa & Sperduti, 2014) are different.

Table 4
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.

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 (Boulanger-Lewandowski et al., 2012)	28.92%	34.02%

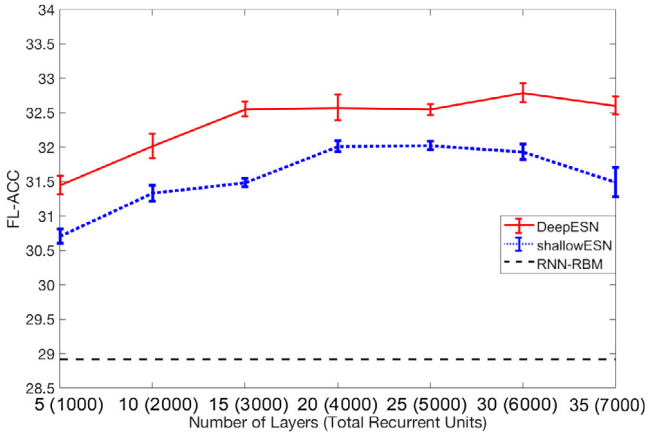

Fig. 6. Comparison between the FL-ACCs (and standard deviations on reservoir guesses represented by vertical intervals) obtained on the validation set of the Piano-midi.de task by DeepESN and shallowESN, considering a number of recurrent layers in the range 5–35 and, correspondingly, a total number of 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 Boulanger-Lewandowski et al. (2012).

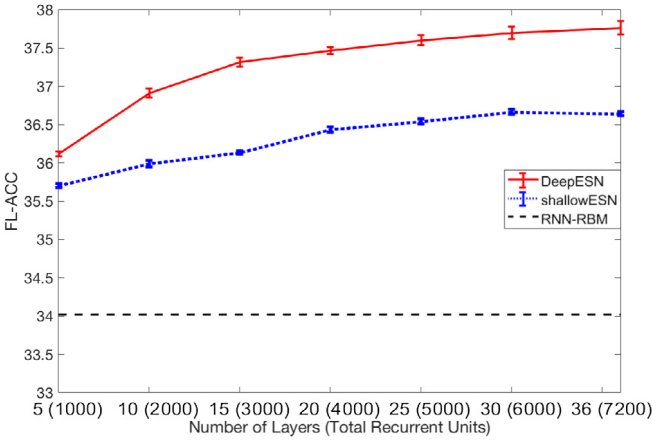

Fig. 7. Comparison between the FL-ACCs (and standard deviations on reservoir guesses represented by vertical intervals) obtained on the validation set of the MuseData task by DeepESN and shallowESN, considering a number of recurrent layers in the range 5–36 and, correspondingly, a total number of recurrent 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 Boulanger-Lewandowski et al. (2012).

on the validation set considering values of the hyper-parameters chosen from the ranges defined in Table 1.

Figs. 6 and 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 Boulanger-Lewandowski et al. (2012) as a horizontal dashed line. As we can see from both Figs. 6 and 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 Figs. 6 and 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.

4.2. Speech recognition

In this section, we consider the isolated spoken digit recognition task discussed in Verstraeten, Schrauwen, Stroobandt, and Van Campenhout (2005). This is a widely used task in the RC context (see e.g. Rodan & Tiňo, 2011, 2012; Verstraeten et al., 2007), 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 (Rodan & Tiňo, 2011, 2012; Verstraeten et al., 2007), 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 Verstraeten et al. (2007), the speech audio was preprocessed using a biological model of the human cochlea by Lyon (1982), resulting in a 77-channel cochleagram.

For what regards DeepESN, the model selection process is performed as in Section 4.1. 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.

Table 5 shows the test WER obtained by DeepESN designed with the proposed design algorithm, shallowESN, Simple Circular Reservoir (SCR) (Rodan & Tiňo, 2011) and Circular Reservoir with Jumps (CRJ) (Rodan & Tiňo, 2012). The hyper-parameters that obtained the best performance in validation set are $N_L = 11$, $N_R = 50$, $\rho = 0.7$, $scale_{in} = 10$, $a = 0.1$, $\sigma_{IP} = 0.1$ and $\lambda_r = 10^{-1}$ for DeepESN and $N_L = 1$, $N_R = 250$, $\rho = 1$, $scale_{in} = 10$, $a = 0.1$, $\sigma_{IP} = 0.1$ and $\lambda_r = 10^{-3}$ for shallow ESN. As results show, on this task the

Table 5
Test WER (and standard deviation on folds, shown in parentheses) obtained on the speech recognition task by DeepESN, shallowESN, SRC and CRJ.

Model	Test WER
DeepESN	**0.0028** (**0.0005**)
shallowESN	0.0530 (0.0318)
SRC (Rodan & Tiňo, 2011)	0.0081 (0.0022)
CRJ (Rodan & Tiňo, 2012)	0.0046 (0.0021)

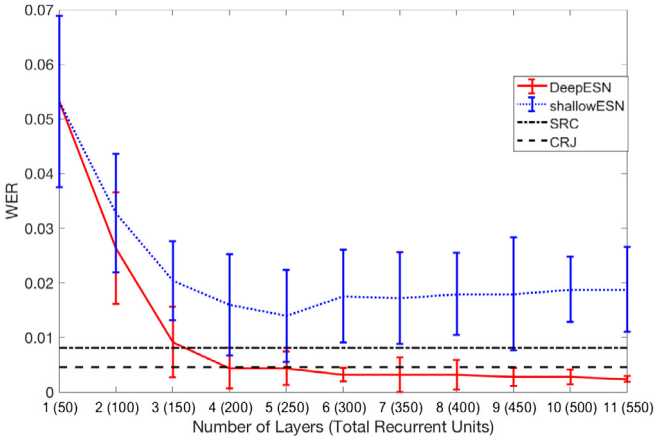

Fig. 8. Comparison of WERs (and standard deviations on folds represented by vertical intervals) obtained on the validation set by DeepESN and shallowESN considering a number of recurrent layer in the range 1–11 and a total number of recurrent units in the range 50–550 on speech recognition task (results 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 Rodan and Tiňo (2012).

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 Fig. 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 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.

Fig. 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.

From Fig. 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 Fig. 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. Discussion

The fundamental goal of the work presented in this paper 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 shallow ESN counterparts, and on the other hand outperform previous state-of-the-art results achieved by fully trained RNN on real-world problems. It is worth to note also that such tasks are characterized by hundreds of sequences with high-dimensionality and very heterogeneous lengths in which the regularization can play a relevant role in relation to our opportunistic and parsimonious construction.

From the point of view of filtering, unlike previous works (Gallicchio, Micheli, & Pedrelli, 2017; Gallicchio et al., 2019), 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 (as observed in previous analyses discussed in Section 1), but also as high pass filter (see Appendix). 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.

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 (Glorot & Bengio, 2010), especially for studies regarding RNNs with ESN-based initialization, see Sutskever, Martens, Dahl, and Hinton (2013). 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 (Jacob, 2004; Saxe, McClelland, & Ganguli, 2013).

Investigations conducted in this paper 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 (Goodfellow et al., 2016; Graves et al., 2013; Pascanu

et al., 2014; Zhang et al., 2016), on the one hand, and between untrained (randomized) and trained RNNs, on the other.

As regards the former question, in this paper, 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 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 paper, 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.

As regards the concrete impact of supervised training of the recurrent connections in RNN applications (the second question above), the results of our experiments on the polyphonic music tasks showed that ESN-based approaches (both in the shallow and in the deep cases) are actually able to outperform fully trained complex RNN architectures that achieved previous state-of-the-art results (as represented, in this case, by the RNN-RBM model). Moreover, the good results obtained by DeepESNs on the isolated-word speech recognition task, also allow to foresee future comparisons between the untrained RC approach and trained gated RNNs, such as Long Short Term Memory (Hochreiter & Schmidhuber, 1997) and Gated Recurrent Unit (Cho et al., 2014), popularized in the context of continuous (rather than isolated-word) speech recognition problems (Graves et al., 2013; Graves & Schmidhuber, 2005), although such analysis would necessarily take into account also other aspects of model design (e.g. bi-directionality or the use of ad-hoc loss functions) that deserve further considerations out the scopes of this paper.

6. Conclusions

In this work, we have proposed 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 turn, 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 assess the effect of the diversification of the frequency components enriching the state representation through layering, we have first 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. After that, we have assessed our approach on challenging real-world tasks in the areas of music and speech processing.

The results achieved on the considered tasks showed that DeepESNs designed by our automatic algorithm consistently improve the performance of shallow ESNs counterparts under the same experimental settings (and ranges for the hyper-parameter values). Noteworthy, the performance achieved by DeepESN outperforms 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 this respect, at the best of our knowledge, the results presented in this paper represent also the first experimental evidence that RC networks with hierarchical reservoir organization consistently outperform RC shallow (one recurrent layer) architectures in challenging real-world tasks.

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.

Acknowledgments

We wish to thank the anonymous action editor and reviewers of this journal for their valuable suggestions and constructive feedbacks, which allowed us to definitely improve the quality of this paper.

Appendix. Experimental analysis of depth on real-world tasks

In this section, we empirically evaluate how the spectral centroid of reservoir states, computed by Algorithm 1 on the layers of the DeepESN, varies with the depth of the stacked recurrent architecture on the considered real-world tasks (see Section 3.4 for methodological details). The aim is to assess the considered η value as well as to evaluate the effectiveness of the Algorithm 2 and the stop condition defined in Eq. (9) on the considered real-world tasks.

Fig. A.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 3.4), the layers 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 centroid after convergence. From this, we can observe that also for this task the value of $\eta = 0.01$ for the stop condition in Eq. (9) is adequate to reach a point near the convergence.

For what regards the polyphonic music tasks (defined in Section 4.1), Figs. A.10 and A.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 Fig. 5) and speech recognition (in Fig. A.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 paper. However, even in this case, the filtering effect becomes progressively negligible in determining the convergence of the spectral centroid. Moreover, we can note that

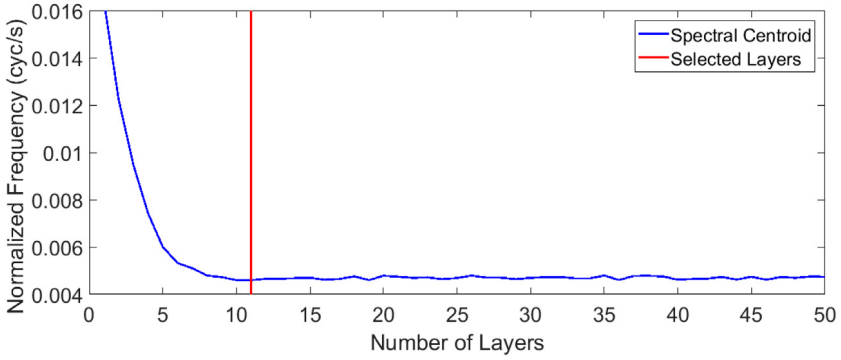

Fig. A.9. Spectral centroid computed on the state of DeepESN layers optimized on speech recognition task. The red vertical line indicates the number of layers selected by the design algorithm. Threshold η value is set to 0.01. Normalized frequency is expressed in cycles per second (cyc/s).

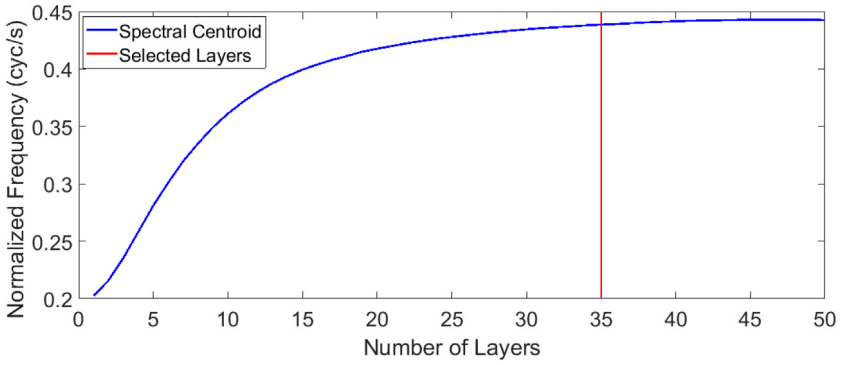

Fig. A.10. Spectral centroid computed on the state of DeepESN layers optimized on Piano-midi.de task. The red vertical line indicates the number of layers selected by the design algorithm. Threshold η value is set to 0.01. Normalized frequency is expressed in cycles per second (cyc/s).

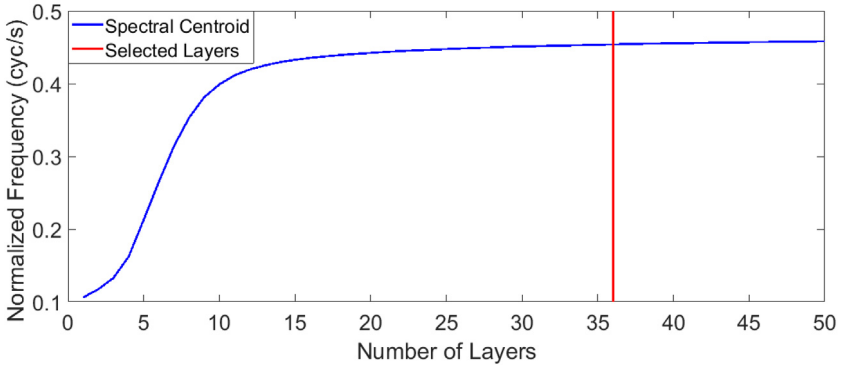

Fig. A.11. Spectral centroid computed on the state of DeepESN layers optimized on MuseData task. The red vertical line indicates the number of layers selected by the design algorithm. Threshold η value is set to 0.01. Normalized frequency is expressed in cycles per second (cyc/s).

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 (Figs. A.10 and A.11) and the performance (Figs. 6 and 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 4.1). 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 (Eq. (9)) 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.

References

Angelov, P., & Sperduti, A. (2016). Challenges in deep learning. In *Proceedings of the 24th European symposium on artificial neural networks (ESANN)* (pp. 489–495). i6doc.com.

Bay, M., Ehmann, A. F., & Downie, J. S. (2009). Evaluation of multiple-f0 estimation and tracking systems. In *10th International society for music information retrieval conference (ISMIR)*.

Bengio, Y., Boulanger-Lewandowski, N., & Pascanu, R. (2013). Advances in optimizing recurrent networks. In *Acoustics, speech and signal processing (ICASSP), 2013 IEEE international conference on* (pp. 8624–8628). IEEE.

Bertschinger, N., & Natschläger, T. (2004). Real-time computation at the edge of chaos in recurrent neural networks. *Neural Computation*, *16*(7), 1413–1436.

Boulanger-Lewandowski, N., Bengio, Y., & Vincent, P. (2012). Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. In *Proceedings of the 29th international conference on machine learning*.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., & Schwenk, H. et al. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078.

Gallicchio, C. (2018). Short-term memory of Deep RNN. In *Proceedings of the 26th european symposium on artificial neural networks (ESANN)* (pp. 633–638). i6doc.com.

Gallicchio, C., Martin-Guerrero, J., Micheli, A., & Soria-Olivas, E. (2017). Randomized machine learning approaches: Recent Developments and challenges. In *Proceedings of the 25th European symposium on artificial neural networks (ESANN)* (pp. 77–86). i6doc.com.

Gallicchio, C., & Micheli, A. (2011). Architectural and markovian factors of echo state networks. *Neural Networks*, *24*(5), 440–456.

Gallicchio, C., & Micheli, A. (2016). Deep reservoir computing: A critical analysis. In *Proceedings of the 24th European symposium on artificial neural networks (ESANN)* (pp. 497–502). i6doc.com.

Gallicchio, C., & Micheli, A. (2017a). Deep Echo State Network (DeepESN): A Brief Survey. arXiv:1712.04323.

Gallicchio, C., & Micheli, A. (2017). Echo state property of deep reservoir computing networks. *Cognitive Computation*, *9*(3), 337–350. http://dx.doi.org/10.1007/s12559-017-9461-9.

Gallicchio, C., & Micheli, A. (2018). Why layering in recurrent neural networks? A DeepESN survey. In *Proceedings of the 2018 IEEE international joint conference on neural networks (IJCNN)* (pp. 1800–1807). IEEE.

Gallicchio, C., Micheli, A., & Pedrelli, L. (2017). Deep reservoir computing: a critical experimental analysis. *Neurocomputing*, *268*, 87–99. http://dx.doi.org/10.1016/j.neucom.2016.12.089.

Gallicchio, C., Micheli, A., & Pedrelli, L. (2019). Hierarchical temporal representation in linear reservoir computing. In A. Esposito, M. Faundez-Zanuy, F. C. Morabito, & E. Pasero (Eds.), *Neural Advances in Processing Nonlinear Dynamic Signals* (pp. 119–129). Cham: Springer International Publishing, WIRN 2017, http://dx.doi.org/10.1007/978-3-319-95098-3_11. arXiv preprint arXiv:1705.05782.

Gallicchio, C., Micheli, A., & Silvestri, L. (2017). Local lyapunov exponents of deep rnn. In *Proceedings of the 25th european symposium on artificial neural networks (ESANN)* (pp. 559–564). i6doc.com.

Gallicchio, C., Micheli, A., & Silvestri, L. (2018). Local lyapunov exponents of Deep Echo State Networks. *Neurocomputing*, *298*, 34–45.

Gallicchio, C., Micheli, A., & Tiňo, P. (2018). Randomized recurrent neural networks. In *Proceedings of the 26th European symposium on artificial neural networks (ESANN)* (pp. 415–424). i6doc.com.

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (pp. 249–256).

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press, http://www.deeplearningbook.org.

Graves, A., Mohamed, A.-R., & Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (Icassp), 2013 Ieee international conference on* (pp. 6645–6649). IEEE.

Graves, A., & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional lstm networks. In *Proceedings of the 2005 IEEE international joint conference on neural networks (IJCNN), vol. 4* (pp. 2047–2052). IEEE.

Hermans, M., & Schrauwen, B. (2013). Training and analysing deep recurrent neural networks. In *Advances in neural information processing systems* (pp. 190–198).

Hihi, S. E., & Bengio, Y. (1995). Hierarchical recurrent neural networks for long-term dependencies. In *Advances in neural information processing systems* (pp. 493–499).

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Holzmann, G., & Hauser, H. (2010). Echo state networks with filter neurons and a delay&sum readout. *Neural Networks*, *23*(2), 244–256.

Jacob, J. M. (2004). *Advanced AC circuits and electronics: principles & applications*. Cengage Learning.

Jaeger, H. (2001). *Short term memory in echo state networks, Tech. rep.*. German National Research Center for Information Technology.

Jaeger, H. (2001). *The "echo state" approach to analysing and training recurrent neural networks - with an erratum note, Tech. rep.,*. GMD - German National Research Institute for Computer Science, Tech. Rep..

Jaeger, H. (2007). *Discovering multiscale dynamical features with hierarchical echo state networks, Tech. rep.*. Jacobs University Bremen.

Jaeger, H., & Haas, H. (2004). Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, *304*(5667), 78–80.

Jaeger, H., Lukoševičius, M., Popovici, D., & Siewert, U. (2007). Optimization and applications of echo state networks with leaky-integrator neurons. *Neural Networks*, *20*(3), 335–352.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097–1105).

Lukoševičius, M. (2012). A practical guide to applying echo state networks. In *Neural networks: Tricks of the trade* (pp. 659–686). Springer.

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

Lyon, R. (1982). 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* (pp. 1282–1285). IEEE.

Maass, R. L. W. (2007). Edge of chaos and prediction of computational performance for neural circuit models. *Neural Networks*, *20*(3), 323–334.

Malik, Z., Hussain, A., & Wu, Q. (2017). Multilayered echo state machine: a novel architecture and algorithm. *IEEE Transactions on Cybernetics*, *47*(4), 946–959.

Mohamed, A.-R., Dahl, G. E., & Hinton, G. (2012). Acoustic modeling using deep belief networks. *IEEE Transactions on Audio, Speech and Language Processing*, *20*(1), 14–22.

Pasa, L., & Sperduti, A. (2014). Pre-training of recurrent neural networks via linear autoencoders. In *Advances in neural information processing systems* (pp. 3572–3580).

Pascanu, R., Gülçehre, Ç., Cho, K., & Bengio, Y. (2014). How to construct deep recurrent neural networks. In *Proceedings of the second international conference on learning representations*.

Qiao, J., Li, F., Han, H., & Li, W. (2017). Growing echo-state network with multiple subreservoirs. *IEEE Transactions on Neural Networks and Learning Systems*, *28*(2), 391–404.

Rodan, A., & Tiňo, P. (2011). Minimum complexity echo state network. *IEEE Transactions on Neural Networks*, *22*(1), 131–144.

Rodan, A., & Tiňo, P. (2012). Simple deterministically constructed cycle reservoirs with regular jumps. *Neural Computation*, *24*(7), 1822–1852.

Saxe, A. M., McClelland, J. L., & Ganguli, S. (2013). Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. arXiv preprint arXiv:1312.6120.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, *61*, 85–117.

Schrauwen, B., Wardermann, M., Verstraeten, D., Steil, J. J., & Stroobandt, D. (2008). Improving reservoirs using intrinsic plasticity. *Neurocomputing*, *71*(7), 1159–1171.

Steil, J. J. (2007). Online reservoir adaptation by intrinsic plasticity for backpropagation–decorrelation and echo state learning. *Neural Networks*, *20*(3), 353–364.

Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *International conference on machine learning* (pp. 1139–1147).

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems* (pp. 3104–3112).

Tiňo, P., Hammer, B., & Bodén, M. (2007). Markovian bias of neural-based architectures with feedback connections. In *Perspectives of neural-symbolic integration* (pp. 95–133). Springer.

Triefenbach, F., Jalalvand, A., Demuynck, K., & Martens, J.-P. (2013). Acoustic modeling with hierarchical reservoirs. *IEEE Transactions on Audio, Speech and Language Processing*, *21*(11), 2439–2450.

Triesch, J. (2005). A gradient rule for the plasticity of a neurons intrinsic excitability. In *International conference on artificial neural networks* (pp. 65–70). Springer.

Verstraeten, D., Schrauwen, B., d'Haene, M., & Stroobandt, D. (2007). An experimental unification of reservoir computing methods. *Neural Networks*, *20*(3), 391–403.

Verstraeten, D., Schrauwen, B., Stroobandt, D., & Van Campenhout, J. (2005). Isolated word recognition with the liquid state machine: a case study. *Information Processing Letters*, *95*(6), 521–528.

Wyffels, F., Schrauwen, B., Verstraeten, D., & Stroobandt, D. (2008). Band-pass reservoir computing. In *Proceedings of the 2008 IEEE international joint conference on neural networks (IJCNN)* (pp. 3204–3209). IEEE.

Xue, Y., Yang, L., & Haykin, S. (2007). Decoupled echo state networks with lateral inhibition. *Neural Networks*, *20*(3), 365–376.

Yamashita, Y., & Tani, J. (2008). Emergence of functional hierarchy in a multiple timescale neural network model: a humanoid robot experiment. *PLoS Computational Biology*, *4*(11), e1000220.

Yildiz, I. B., Jaeger, H., & Kiebel, S. (2012). Re-visiting the echo state property. *Neural Networks*, *35*, 1–9.

Zhang, S., Wu, Y., Che, T., Lin, Z., Memisevic, R., Salakhutdinov, R. R., et al. (2016). Architectural complexity measures of recurrent neural networks. In *Advances in Neural Information Processing Systems* (pp. 1822–1830).