Tree Echo State Networks

Claudio Gallicchio[a], Alessio Micheli[a,*]

[a]*Department of Computer Science, University of Pisa, Largo B. Pontecorvo 3, 56127 Pisa, Italy*

Abstract

In this paper we present the Tree Echo State Network (TreeESN) model, generalizing the paradigm of Reservoir Computing to tree structured data. TreeESNs exploit an untrained generalized recursive reservoir, exhibiting extreme efficiency for learning in structured domains. In addition, we highlight through the paper other characteristics of the approach: First, we discuss the Markovian characterization of reservoir dynamics, extended to the case of tree domains, that is implied by the contractive setting of the TreeESN state transition function. Second, we study two types of state mapping functions to map the tree structured state of TreeESN into a fixed-size feature representation for classification or regression tasks. The critical role of the relation between the choice of the state mapping function and the Markovian characterization of the task is analyzed and experimentally investigated on both artificial and real-world tasks. Finally, experimental results on benchmark and real-world tasks show that the TreeESN approach, in spite of its efficiency, can achieve comparable results with state-of-the-art, althought more complex, neural and kernel based models for tree structured data.

Keywords: Reservoir Computing, Echo State Networks, Learning in Tree Structured Domains

Reference to Published Article
C. Gallicchio, A. Micheli, Tree Echo State Networks, Neurocomputing, vol. 101, pp. 319-337, Elsevier, DOI: 10.1016/j.neucom.2012.08.017, ISSN: 0925-2312, 2013.

*Corresponding author. Tel.: +39 050 2212798.
Email addresses: `gallicch@di.unipi.it` (Claudio Gallicchio), `micheli@di.unipi.it` (Alessio Micheli)

1. Introduction

Traditional Machine Learning models, including neural networks, are suitable for dealing with flat data. Thereby, when applied to real-world problems, they often need to resort to fixed-size vectorial representations (or descriptors) of the input data under consideration. This involves a number of undesirable drawbacks such as the possible loss of relational information in the original data and the need of domain experts to design appropriate descriptors for the problem at hand. Variable size structures, such as sequences and trees, often represent more suitable means for describing relations among the entities involved in many real-world applicative domains, including Chemistry, Molecular Biology, Automated Reasoning and Speech and Text Processing. It is therefore conceivable that the generalization of Machine Learning for directly processing structured information represents a wide area of research including very different paradigms such as inductive logic programming and statistical relational learning (see, e.g., [1] and [2]), relational data mining (see, e.g., [3]), tree-distance based approaches (see, e.g., [4, 5, 6]), probabilistic learning (see, e.g. [7, 8, 9], and more recently [10, 11]), neural networks and kernel-based approaches. In particular, we focus on the latter neurocomputing related approaches, i.e. neural and kernel based models, for supervised learning (classification or regression) on tree structured domains, which has increasingly attracted research interest during the last two decades. However, the design of such models for treating structured information has raised several research issues [12]. In this concern, the efficiency of learning algorithms deserves particular attention from a theoretical and practical point of view.

Recurrent Neural Networks (RNNs) represent a widely known and used class of neural network models for learning in sequential domains. In this context, Reservoir Computing (RC) [13, 14] models in general, and Echo State Networks (ESNs) [15, 16] in particular, are becoming more and more popular as extremely efficient approaches for RNN modeling. An ESN consists in a large non-linear *reservoir* hidden layer of sparsely connected recurrent units and a linear *readout* layer. The readout is the only trained part of the architecture, whereas the reservoir is initialized to satisfy a condition of the state dynamics called the *Echo State Property* (ESP) [15] and then is left untrained. A contractive initialization of state transition functions in RNNs implies a bias on the state dynamics towards Markovian process modeling. This property leads to the ability of inherent discrimination among different input histories even prior to training [17, 18]. Through the use of fixed contractive state dynamics this ability is efficiently exploited by ESNs [19, 20].

Recursive Neural Networks (RecNNs) [21, 7] represent a generalization of RNNs for processing of hierarchical data structures (e.g. rooted trees) constituting a powerful class of models [22, 23] with applications in many real-world domains (e.g. [24, 25, 26, 27, 28, 29]). Recent developments concerned models such as Relational Neural Networks (RelNNs)[30, 31] and Graph Neural Networks (GNNs) [32]. In particular, GNN is a recursive neural network based on a constraint of contractivity applied to the learning of the state transition function. As such, once applied to the same tree structured data, these models allow us to have a direct comparison of TreeESN with the class of RecNNs. However, the known issues related to RNN training (e.g. [14, 33]) continue to hold also for RecNNs [34], in which case training may be even more computationally expensive. Another widely used class of learning models for tree structured domains is represented by the kernel methods for trees (see e.g. [35, 12]), among which a common approach is represented by the convolution tree kernels (e.g. [36, 37, 38, 39, 40]). Such approaches, however, are typically very expensive in terms of the required computational cost for the kernel computation (possibly implying a quadratic cost in the number of input nodes) and for training the Support Vector Machine (SVM). Thereby, the investigation of possible efficient approaches for the adaptive processing of tree structured data represents a very appealing and worth of interest open research problem.

In this paper we present the Tree Echo State Network (TreeESN) model[1] to extend the applicability of the ESN approach to tree structured data and to allow for an extremely efficient modeling of RecNNs. As for standard ESNs, the architecture of a TreeESN is composed of an untrained recurrent non-linear reservoir and a linear readout that can be trained by efficient linear methods. In a TreeESN input trees are processed in a bottom-up recursive fashion, from leaves to the root by the application of the generalized reservoir architecture to each node, and hence building a structured state representation. For modeling functions in which input structures are

[1]A short preliminary version of the paper was recently proposed in [41]. Here we extend the work in [41] through an in-dept study of Markovianity for trees, the introduction of the computational complexity analysis and new experiments.

mapped into unstructured output vectors (e.g. for classification or regression tasks on trees), the structured state computed by the reservoir can be mapped into a fixed-size feature representation according to different *state mapping functions*. A contractive setting of the reservoir dynamics is inherited from ESN for sequences. Beside ensuring stability of state dynamics, contractivity allows us also to study an interesting region of the state space of recursive models characterized by Markovian nature, generalized to the case of tree structure processing [34]. Moreover, based on the ability of inherently discriminating among input structures in absence of learning of recurrent connections, TreeESNs represent both an architectural and an experimental performance baseline for RecNN models with trained recurrent dynamics.
Although the framework of recursive structural transduction is not new, its modeling through reservoir computing introduces interesting advantage concerning the efficiency. It also opens experimental challenging issues in terms of the effectiveness in comparison to more complex approaches and in terms of the evaluation of the effects of Markovianity and of state mapping functions. In particular, the role of the state mapping function in relation to Markovianity has a relevant impact on the organization of the feature space and on the predictive performance of TreeESNs, that we investigate through experiments on artificial and real-world tasks. The effective evaluation of the efficiency and of the predictive performance is investigated through a set of variegate tasks using different performance measures according to the literature results.

This paper is organized as follows. Section 2 reviews the foundations for recursive processing of tree structured data[2]. The core of the ESN for trees (TreeESN) is presented in Section 3, describing the novelties concerning the architecture, the computational complexity (efficiency) of the model, the initialization conditions and the Markovian characterization of reservoir dynamics. Section 4 presents experimental applications of TreeESNs on tasks on tree domains of different nature. First, we analyze the characteristics and limitations of the approach related to the state mapping function and Markovianity. Then, we show the potentiality of the model in terms of efficiency and predictive performance through comparisons with related approaches in

[2]Section 2 is useful to introduce readers to transductions on trees, which is new in the context of Reservoir Computing for sequences. Readers familiar with recursive neural models for structured data can skip this section except for the notation and the introduction of the *state mapping function* $\chi()$ (equation 9).

the neural and kernel field. Finally, conclusions are discussed in Section 5.

2. Recursive Transductions on Tree Domains

In this Section we review the framework of transductions on trees (i.e. hierarchical discrete structured data). The concept of structural transduction and the extension of the input domain are the ground to extend Reservoir Computing approaches for sequence (signal/series processes) to discrete hierarchical processing. Although new in the context of Reservoir Computing, the description in this section is rooted in the previous framework for RecNN approaches for both supervised [21, 7, 23] and unsupervised learning [42, 43].

2.1. Tree Domains

In this paper we are interested in processing models on tree structured domains. In particular, we will restrict our focus on *rooted* trees, and the word rooted will therefore be omitted for the sake of simplicity. A tree is represented by $\mathbf{t}$ and its set of nodes is denoted by $N(\mathbf{t})$, whereas $|N(\mathbf{t})|$ is the number of nodes in $\mathbf{t}$. The root of $\mathbf{t}$ is denoted by $root(\mathbf{t})$, while a generic node in $N(\mathbf{t})$ is indicated by n. The i-th child of n is denoted by $ch_i(n)$, whereas the number of children of n is the *degree* of n. A k-ary tree is a tree in which every node has degree k. In the following, the symbol k is used to refer the maximum *degree* over a set of trees of interest. In general, a k-ary tree is positional, i.e. for each node n it is possible to distinguish among the positions of its children. In a non positional tree, the children of each node n can be enumerated, but their positions cannot be distinguished.

The *ancestors* of a node n are the nodes in the unique path from $root(\mathbf{t})$ to n. If v is an ancestor of n we also say that n is a *descendant* of v. We denote by $\mathbf{t}(n)$ the *sub-tree rooted at* n, i.e. the tree induced by the descendants of n and rooted at n. The *depth* of a node n in $\mathbf{t}$, i.e. $depth(n, \mathbf{t})$, is the length of the path from $root(\mathbf{t})$ to n. The largest among the depths of the nodes in $\mathbf{t}$ is the *height* of $\mathbf{t}$, denoted as $h(\mathbf{t})$.

In a recursive definitions of trees, a k-ary tree $\mathbf{t}$ can be defined as either the *empty tree*, denoted by *nil*, or as the root node n and the k sub-trees rooted in its children, denoted by $n(\mathbf{t}(ch_1(n)), \ldots, \mathbf{t}(ch_k(n)))$. Note that in this recursive definition, some of the k children of n could be *absent* (or *missing*), in which case the corresponding sub-trees are empty.

The suffix of height $h \geq 0$ of a tree $\mathbf{t}$, indicated as $S_h(\mathbf{t})$, is the tree obtained from $\mathbf{t}$ by removing every node whose depth is greater than h:

$$S_h(\mathbf{t}) = \begin{cases} nil & if \;\; h = 0 \;\; or \;\; \mathbf{t} = nil \\ & \\ n(S_{h-1}(\mathbf{t}(ch_1(n))), \ldots, & if \;\; h > 0 \;\; and \\ \quad S_{h-1}(\mathbf{t}(ch_k(n)))) & \mathbf{t} = n(\mathbf{t}(ch_1(n)), \ldots, \mathbf{t}(ch_k(n))) \end{cases} \tag{1}$$

We consider *labeled* trees, in which a (numerical) label information is assigned to every node. The set of trees with node labels in the domain $\mathcal{L}$, is denoted by $\mathcal{L}^{\#}$, while $\mathcal{L}^{\#k}$ is used whenever the maximum degree k is specified. The *skeleton* of a tree $\mathbf{t} \in \mathcal{L}^{\#}$, denoted by $skel(\mathbf{t})$, is the tree obtained from $\mathbf{t}$ by removing the labels on its nodes.

In the following, as label domains of interest we consider the N_U dimensional real input space $\mathbb{R}^{N_U}$ and the N_Y dimensional output space $\mathbb{R}^{N_Y}$. Accordingly, input and output structured domains are denoted by $(\mathbb{R}^{N_U})^{\#}$ and $(\mathbb{R}^{N_Y})^{\#}$, respectively. Moreover, the label associated to node n of an input tree is denoted by $\mathbf{u}(n)$, whereas for an output tree the label of n is indicated by $\mathbf{y}(n)$.

Note that special cases of trees are represented by *sequences*, which can be viewed as 1-ary trees in which the leaf node corresponds to the oldest element and the root to the most recent one.

2.2. Structural Transductions on Tree Domains

We are interested in computing *structural transductions* on trees, i.e. functions for which the input and the output spaces are tree domains:

$$\mathcal{T} : (\mathbb{R}^{N_U})^{\#k} \rightarrow (\mathbb{R}^{N_Y})^{\#k} \tag{2}$$

Such structural transductions can be qualified in different ways. $\mathcal{T}$ is a *tree-to-tree* (or *structure-to-structure*) transduction if its output is always isomorphic to its input, i.e. $\forall \mathbf{t} \in (\mathbb{R}^{N_U})^{\#}, \;\; skel(\mathcal{T}(\mathbf{t})) = skel(\mathbf{t})$. $\mathcal{T}$ is a *tree-to-element* (or *structure-to-element*) transduction if the output computed for each input tree is a single unstructured vector, i.e. a trivial tree with a single node, and the output domain reduces to the fixed-size vectorial space $\mathbb{R}^{N_Y}$. $\mathcal{T}$ is *causal* if the output computed for each node n of an input tree depends only on n and the descendants of n. $\mathcal{T}$ is *stationary* if the mapping applied to each node of the input tree does not depend on the particular node to which it is applied. An *adaptive* transduction is learned from examples, whereas a *fixed* transduction is a-priori defined.

We are in particular interested in the possibility of computing structural transductions based on recursive approaches. To this aim we find it useful to consider tree transductions admitting a *recursive state representation* [7]. In this case, a transduction $\mathcal{T}$ can be usefully decomposed in an *encoding transduction*, denoted by $\mathcal{T}_{enc}$, and an *output transduction*, denoted by $\mathcal{T}_{out}$:

$$\mathcal{T} = \mathcal{T}_{out} \circ \mathcal{T}_{enc} \tag{3}$$

The encoding transduction $\mathcal{T}_{enc}$ is a tree-to-tree transduction that maps an input tree $\mathbf{t}$ into a corresponding tree structured feature representation (or state) $\mathbf{x}(\mathbf{t})$:

$$\begin{aligned} &\mathcal{T}_{enc} : (\mathbb{R}^{N_U})^{\#k} \to (\mathbb{R}^{N_R})^{\#k} \\ &\mathbf{x}(\mathbf{t}) = \mathcal{T}_{enc}(\mathbf{t}) \end{aligned} \tag{4}$$

where $(\mathbb{R}^{N_R})^{\#}$ is the tree structured state space and $\mathbb{R}^{N_R}$ is the N_R dimensional numerical feature label space. In the following, we adopt the assumption of considering causal and stationary encoding transductions, which can be computed by resorting to a *node-wise encoding function* τ:

$$\begin{aligned} &\tau : \mathbb{R}^{N_U} \times \mathbb{R}^{kN_R} \to \mathbb{R}^{N_R} \\ &\mathbf{x}(n) = \tau(\mathbf{u}(n), \mathbf{x}(ch_1(n)), \ldots, \mathbf{x}(ch_k(n))) \end{aligned} \tag{5}$$

where $\mathbf{x}(n)$ is the state representation computed for node n and $\mathbf{x}(ch_1(n)), \ldots,$ $\mathbf{x}(ch_k(n))$ are the states associated to the children of n. Equation 5 describes the relation between the state corresponding to a node n and the states corresponding to its children. Note that if one of the children of n is absent then a null state is used for it, e.g. $\mathbf{x}_{nil} = \mathbf{0} \in \mathbb{R}^{N_R}$. The node-wise encoding function τ of equation 5 induces a recursive function on trees, denoted by $\hat{\tau}$

$$\begin{aligned} &\hat{\tau} : (\mathbb{R}^{N_U})^{\#k} \times \mathbb{R}^{N_R} \to \mathbb{R}^{N_R} \\ &\hat{\tau}(\mathbf{t}, \mathbf{x}_{nil}) = \begin{cases} \mathbf{x}_{nil} & if \;\; \mathbf{t} = nil \\ \begin{aligned} &\tau(\mathbf{u}(n), \hat{\tau}(\mathbf{t}(ch_1(n)), \mathbf{x}_{nil}), \ldots, \\ &\quad \hat{\tau}(\mathbf{t}(ch_k(n)), \mathbf{x}_{nil}))) \end{aligned} & if \;\; \mathbf{t} = n(\mathbf{t}(ch_1(n)), \ldots, \mathbf{t}(ch_k(n))) \end{cases} \\ &\mathbf{x}(root(\mathbf{t})) = \hat{\tau}(\mathbf{t}, \mathbf{x}_{nil}) \end{aligned} \tag{6}$$

where $\hat{\tau}(\mathbf{t}, \mathbf{x}_{nil})$ is the state of the root of $\mathbf{t}$, i.e. $\mathbf{x}(root(\mathbf{t}))$, given that the null state for absent nodes is $\mathbf{x}_{nil}$ (which is used as base for the recursive definition of $\hat{\tau}$). The recursive definition of $\hat{\tau}$ is given in equation 6 in analogy with the extension to paths of the definition of transition functions in finite automata for sequences [44]. Equation 6 describes the relation between the state of a node n and the states computed for the descendants of n. Thus, the computation of the structured feature representation corresponding to an input tree $\mathbf{t}$, i.e. $\mathcal{T}_{enc}(\mathbf{t})$, consists in the application of $\hat{\tau}$ to $\mathbf{t}$. This implies the computation of the state for each node n in $\mathbf{t}$ through the application of the node-wise encoding function τ according to a bottom-up visit of $\mathbf{t}$ (i.e. starting from the leaf nodes and ending in the root). According to the assumption of stationarity, function τ of equation 5 is applied in correspondence of every visited node n, taking as inputs the label of n and the states already computed for the children of n. This bottom-up recursive encoding process is shown in Figure 1 through two illustrative examples.

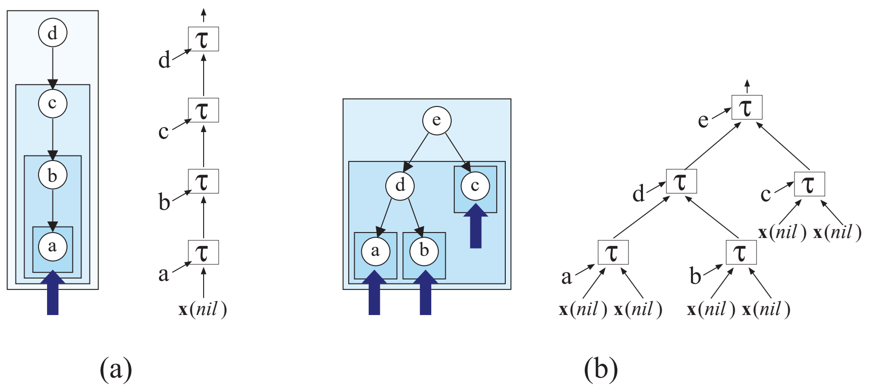


Figure 1: Bottom-up recursive encoding process on trees. Example (a) shows the process of visit for the special case of sequential input (i.e. $k = 1$), while example (b) shows the same process for the case of binary trees (i.e. $k = 2$).

The output transduction $\mathcal{T}_{out}$ is then used to map the structured state representation of an input tree into its corresponding output.

For tree-to-tree transductions $\mathcal{T}$, the output transduction $\mathcal{T}_{out}$ is also tree-to-tree:

$$\begin{aligned} \mathcal{T}_{out} &: (\mathbb{R}^{N_R})^{\#k} \rightarrow (\mathbb{R}^{N_Y})^{\#k} \\ \mathbf{y}(\mathbf{t}) &= \mathcal{T}_{out}(\mathbf{x}(\mathbf{t})) \end{aligned} \tag{7}$$

where $\mathbf{y}(\mathbf{t})$ represents the structured output associated to $\mathbf{t}$. $\mathcal{T}_{out}$ can be computed by resorting to a *node-wise output function* g:

$$\begin{aligned} &g : \mathbb{R}^{N_R} \to \mathbb{R}^{N_Y} \\ &\mathbf{y}(n) = g(\mathbf{x}(n)) \end{aligned} \tag{8}$$

where $\mathbf{y}(n) \in \mathbb{R}^{N_Y}$ is the (unstructured) output associated to node n. In this case $\mathbf{y}(\mathbf{t})$ is computed by applying function g to every node of $\mathbf{t}$. Figure 2 summarizes the computation of tree-to-tree structural transductions.

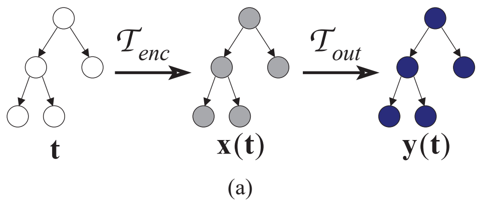


Figure 2: Computation of a tree-to-tree structural transduction $\mathcal{T}$.

For tree-to-element transductions, a *state mapping function* χ is preliminarly applied to the structured state computed by $\mathcal{T}_{enc}$, in order to obtain a single *fixed-size* feature state, representative of the whole input structure:

$$\chi : (\mathbb{R}^{N_R})^{\#k} \to \mathbb{R}^{N_R} \tag{9}$$

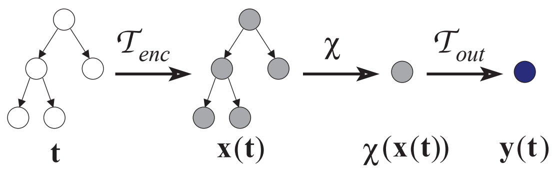
The unstructured fixed-size output is then computed by applying the node-wise output function g of equation 8 only to the output of the state mapping function:

$$\mathbf{y}(\mathbf{t}) = g(\chi(\mathbf{x}(\mathbf{t}))) \tag{10}$$

In this case, $\mathbf{y}(\mathbf{t}) \in \mathbb{R}^{N_Y}$ denotes the unstructured output associated to $\mathbf{t}$. Moreover, input and output domains of $\mathcal{T}_{out}$ of equation 7 actually degenerate into the unstructured vectorial spaces $\mathbb{R}^{N_R}$ and $\mathbb{R}^{N_Y}$ respectively, and the computation of the whole transduction can be expressed as the composition of the encoding transduction, the state mapping function and the output transduction:

$$\mathcal{T} = \mathcal{T}_{out} \circ \chi \circ \mathcal{T}_{enc} \tag{11}$$

Figure 3 graphically shows the steps of the computation of a tree-to-element structural transduction.



(b)

Figure 3: Computation of a tree-to-element structural transduction $\mathcal{T}$.

Considering a supervised learning paradigm, a training set of input trees with maximum degree k is represented by $\mathscr{T} = \{(\mathbf{t}, \mathbf{y}_{target}(\mathbf{t})) : \mathbf{t} \in (\mathbb{R}^{N_U})^{\#k}$, $\mathbf{y}_{target}(\mathbf{t}) \in (\mathbb{R}^{N_Y})^{\#k}\}$, where $\mathbf{y}_{target}(\mathbf{t})$ is the target output associated to tree $\mathbf{t}$ in the general case of a tree structured output (tree-to-tree transduction). In the following of the paper, we focus on tree-to-element transductions which allow us an immediate assessing of the TreeESN (provided with different state mapping functions) on known problems modeling a variety of interesting tasks on tree domains, such are classifications of trees. For the specific case of regression tasks related to tree-to-element transductions, $\mathbf{y}_{target}(\mathbf{t})$ is a fixed size vector in $\mathbb{R}^{N_Y}$. In the case of classification tasks with M target classes, $\mathbf{y}_{target}(\mathbf{t})$ is a M-dimensional binary vector in which, for the instances used in the experiments, the element corresponding to the correct classification is equal to 1 and all the other elements are equal to 0.

2.3. RecNNs and Related Models for Processing Structural Trasductions

When the node-wise encoding and output functions of a structural transduction, i.e. τ and g in the above definitions (equations 5 and 8, respectively) are computed by neural networks, we get RecNN models [21, 7]. In the simplest architectural setting, a RecNN consists in a recursive hidden layer, which is responsible for the computation of τ, and a feed-forward output layer, which is responsible for the computation of g. Structural transductions computed by RecNNs are usually characterized by causality, stationarity and adaptivity, as the parameters of both the hidden and the output layers are trained from examples. RecNNs have been sucessfully applied to several real-world applicative domains of relevant interest, including Cheminformatics (e.g. [25, 45, 29]), Natural Language Processing [26, 28] and Image Analysis [24, 27]. In addition, a number of results concerning the computational capabilities of RecNN models have been established, including universal approximation theorems for tree domains processing [23, 22].

The recursive dynamics can also be exploited for unsupervised learning, as introduced in [46] using self-organizing maps, or more in general in [42, 43]. A recent RecNNs variant is represented by the RelNN model [30, 31], in which encoding is based on a sequential processing of the children of each node. The GNN model [32], based on a trained encoding process under contractive constraints, extends the recursive approaches allowing cyclic dynamics in the definition of the state to process graph structures (and hence also tree structures).

However, training RecNNs can face similar problems to those encountered with RNNs [14, 33, 34], such as high computational training costs, local minima, slow convergence and vanishing of the gradients [47]. In particular, training RecNNs can be even more computationally expensive than training RNNs. In this regard, the RC paradigm represents a natural candidate for investigating efficient approaches to RecNN modeling. The following Section 3 describes the TreeESN model, a first extension of the RC paradigm to structured domains processing.

3. TreeESN Model

TreeESNs are RecNNs implementing causal, stationary and partially adaptive transductions on tree structured domains. A TreeESN is composed of an untrained hidden layer of recursive non-linear units (a generalized *reservoir*) and of a trained output layer of feed-forward linear units (the *readout*). The reservoir implements a *fixed* encoding transduction, whereas the readout implements an *adaptive* output transduction. For tree-to-element transductions, a *state mapping function* is used to obtain a single fixed-size feature representation. The following sub-sections describe the components of a TreeESN model.

3.1. Reservoir of TreeESN

The reservoir consists in N_R recursive (typically) non-linear units, which are responsible for computing the encoding of a tree transduction by implementing the node-wise encoding function τ of equation 5, which takes the role of a recursive *state transition function.* Accordingly, the state corresponding to node n of a tree $\mathbf{t}$ is computed as follows:

$$\mathbf{x}(n) = f(\mathbf{W}_{in}\mathbf{u}(n) + \sum_{i=1}^{k} \hat{\mathbf{W}}\mathbf{x}(ch_i(n))) \tag{12}$$

where $\mathbf{W}_{in} \in \mathbb{R}^{N_R \times N_U}$ is the input-to-reservoir weight matrix (which might also contain a bias term), $\hat{\mathbf{W}} \in \mathbb{R}^{N_R \times N_R}$ is the recurrent reservoir weight matrix and f is the element-wise applied activation function of the reservoir units (in this paper we use $tanh$). In correspondence of absent children of node n, a null state $\mathbf{x}_{nil} = \mathbf{0} \in \mathbb{R}^{N_R}$ is used. As in standard ESNs, a sparse pattern of connectivity among the reservoir units is used, i.e. $\hat{\mathbf{W}}$ is a sparse matrix. Notice that if the input structures reduce to sequences, i.e. for $k = 1$, the generalized reservoir of a TreeESN (whose dynamics are described in equation 12) reduces to a standard reservoir of an ESN.

Equation 12 is customized for non-positional trees, whereas in the case of positional trees it can be modified so that different recurrent weight matrices are used in correspondence of different child positions.

Figure 4 depicts the application of the generalized reservoir architecture to a node n of an input tree. The reservoir units are fed with an external input

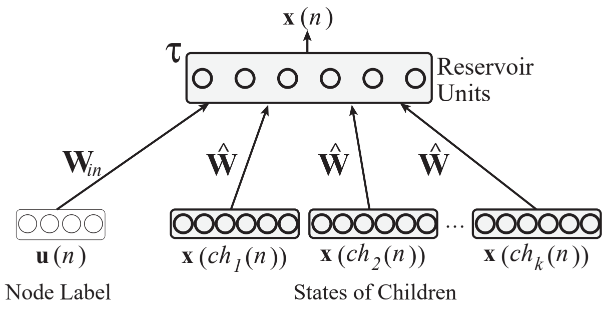


Figure 4: The application of the generalized reservoir architecture of a TreeESN to node n of an input tree.

consisting in the numerical label attached to node n, i.e. $\mathbf{u}(n)$, weighted by the input-to-reservoir weights in $\mathbf{W}_{in}$. Each reservoir unit receives in input also the activation of the reservoir units computed for each child of node n, weighted by the weights in $\hat{\mathbf{W}}$. Note that not every couple of reservoir units is connected, according to the sparse pattern of connectivity within the reservoir (see Figure 5). Moreover, a connection between two reservoir units carries all the corresponding state information computed for the children of the node n. This is illustrated in Figure 5, showing that a connection from unit B to unit A in the generalized reservoir architecture carries to unit A the whole set of activations of unit B computed in correspondence of every child of n, i.e. $x_B(ch_1(n)), \ldots, x_B(ch_k(n))$. Assuming a number of $R <$

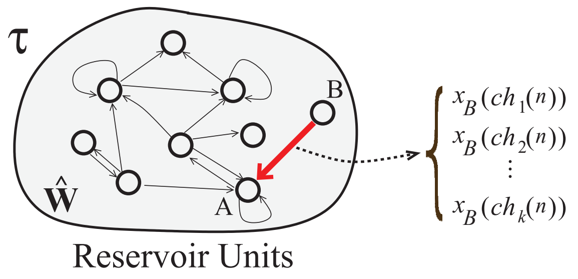


Figure 5: The generalized reservoir architecture of a TreeESN.

N_R recurrent connections for each reservoir unit (sparse connectivity of the reservoir), the total number of recurrent weights in the TreeESN architecture is given by kRN_R. The number of different recurrent reservoir weights then reduces to RN_R because of the assumption of stationarity for processing the states of the children (see equation 12).

The generalized reservoir architecture described in Figure 4 is unfolded on an input tree $\mathbf{t}$ according to the bottom-up recursive encoding process shown in Figure 1 and corresponding to the application to $\mathbf{t}$ of the function $\hat{\tau}$ (equation 6) induced by the reservoir implementation (equation 12) of function τ (equation 5). In practice, the same reservoir architecture is recursively applied to each node of $\mathbf{t}$, following the topology of $\mathbf{t}$ (with an information flow that is isomorphic to the structure of $\mathbf{t}$ itself). An example of this encoding process is shown in Figure 6, in which the same binary tree presented in Figure 1 is considered as input tree $\mathbf{t}$. The discrete input space $\{a, b, c, d, e\}$ is encoded using a 1-of-5 binary encoding such that a is encoded as 10000, b as 01000 and so on up to e encoded as 00001, i.e. $N_U = 5$. The reservoir states for the leaf nodes in $\mathbf{t}$ (i.e. for nodes labeled by a, b and c in Figure 6) are computed first, given the encoding of the corresponding labels and the null state for absent children. The reservoir state for the node with label d is then computed, given the encoding of the corresponding label and the reservoir states computed for the nodes labeled by a and b. Finally, the state for the root of $\mathbf{t}$ (with label e) is computed, given the encoding for its label and the reservoir states computed for the nodes with labels d and c. If a different tree structure is given in input, the encoding process changes according to a topology which is isomorph to the topology of the input structure (see Figure 1). Note that the number of nodes (i.e. $|N(\mathbf{t})|$) and the topology of the input tree are independent of the number of units

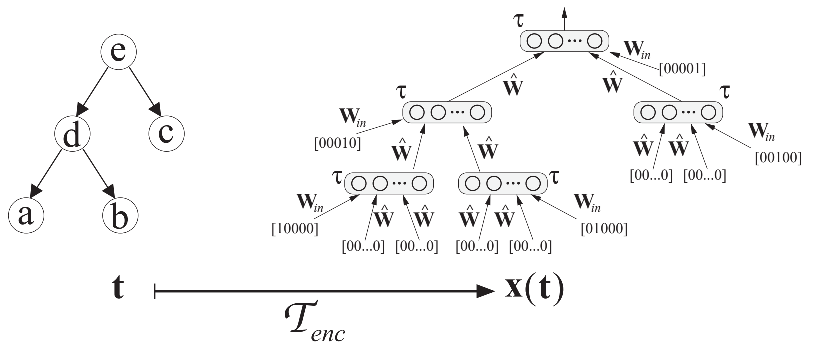


Figure 6: Example of the encoding process computed by the reservoir of a TreeESN. Symbolic node labels are encoded using a 1-of-m binary encoding.

(i.e. N_R) and the topology of the reservoir architecture.

A further point to remark is that, differently from the standard ESN approach (that is specifically designed for signals/time series processing), the reservoir of a TreeESN is run only once on each finite input structure (see Figure 6), without any initial transient to discard. This aspect of reservoir computation is related to the different nature of the data under consideration. In fact, the purpose of reservoir computation in TreeESNs is to encode a set of discrete finite input structures, rather than to build an high dimensional non-linear dynamical representation of a (single, possibly infinite) input signal stream.

As in the standard ESN model, the parameters of the reservoir of a TreeESN are left untrained after initialization. In particular, matrices $\mathbf{W}_{in}$ and $\hat{\mathbf{W}}$ in equation 12 are randomly chosen and then $\hat{\mathbf{W}}$ is scaled to implement a *contractive* state transition function τ. The contractive setting of the reservoir state transition function has the twofold effect of ensuring stability of reservoir dynamics (regardless of other initialization aspects) and of bounding such dynamics into a region of the state space characterized by a Markovian flavor organization [34]. Section 3.5 details the initialization setting and the resulting Markovian characterization of TreeESN reservoir dynamics.

3.2. State Mapping Function: TreeESN-R and TreeESN-M

When processing tree-to-element transductions with variable size (and topology) input trees, the feed-forward readout architecture cannot be applied to the structured reservoir state directly. Such structured state is indeed isomorphic to the variable size input, whereas the number of readout parameters (i.e. the weights in $\mathbf{W}_{out}$) is fixed. Thereby, in order to implement tree-to-element transductions, the structured state representation computed for an input tree $\mathbf{t}$, i.e. $\mathbf{x}(\mathbf{t})$, is mapped into a fixed-size N_R dimensional feature representation through the state mapping function χ of equation 9. In this paper we consider two possible choices for the state mapping function, namely a root state mapping and a mean state mapping.

The *root state mapping* consists in selecting the state of the root of $\mathbf{t}$:

$$\chi(\mathbf{x}(\mathbf{t})) = \mathbf{x}(root(\mathbf{t})) \tag{13}$$

The root state mapping is used in standard RecNNs, in which the state of the root is always used as representative of the whole input structure. Note that when a TreeESN with root state mapping is used to process sequential inputs, the standard ESN model arises.

The *mean state mapping* computes the mean over the states of the nodes in $\mathbf{t}$:

$$\chi(\mathbf{x}(\mathbf{t})) = \frac{1}{|N(\mathbf{t})|} \sum_{n \in N(\mathbf{t})} \mathbf{x}(n) \tag{14}$$

By using the mean state mapping, the fixed-size feature representation $\chi(\mathbf{x}(\mathbf{t}))$ depends (to the same extent) on the state of every node in the input structure $\mathbf{t}$, rather than depending only on the state of a particular node.

In the following, TreeESN-R and TreeESN-M are respectively used to refer to a TreeESN with root state mapping and to a TreeESN with mean state mapping.

3.3. Readout of TreeESN

The readout consists in N_Y feed-forward linear units and is used to process the adaptive output of a tree transduction by implementing the node-wise output function g of equation 8.

For tree-to-tree transductions, the readout is applied to the state of each node n in the input structure:

$$\mathbf{y}(n) = \mathbf{W}_{out}\mathbf{x}(n) \tag{15}$$

where $\mathbf{W}_{out} \in \mathbb{R}^{N_Y \times N_R}$ is the reservoir-to-readout weight matrix (possibly including a bias term). For tree-to-element transductions, the readout is applied only to the output of the state mapping function:

$$\mathbf{y}(\mathbf{t}) = \mathbf{W}_{out}\chi(\mathbf{x}(\mathbf{t})) \tag{16}$$

Figure 7 shows an example of the application of the mean state mapping and of the output function for processing tree-to-element transductions with TreeESNs, referring to the same input tree as in Figure 6.

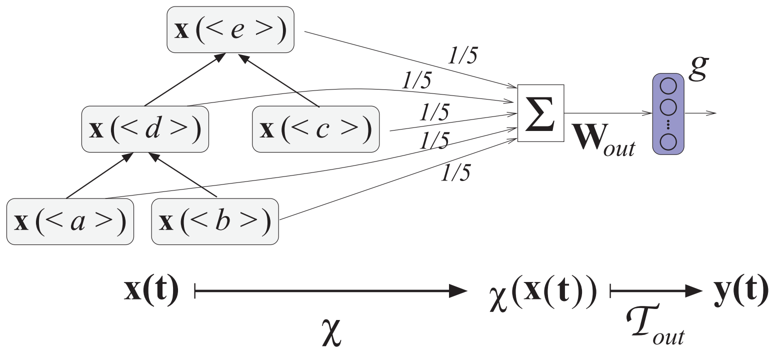


Figure 7: Example of mean state mapping and output functions computed by TreeESNs. The notation $\mathbf{x}(< a >)$ is used here to denote the state computed for the node with label "a" in the input tree.

As for standard ESNs, (off-line) training of the readout is performed by adjusting the weight values in $\mathbf{W}_{out}$ to solve a linear regression problem. Let us consider a training set $\mathscr{T}$ containing a number of P patterns. For tree-to-tree transductions, input patterns in $\mathscr{T}$ correspond to nodes and the corresponding states computed by the reservoir are column-wise arranged into a state matrix $\mathbf{X} \in \mathbb{R}^{N_R \times P}$. Analogously, the target outputs for the patterns in $\mathscr{T}$ are column-wise arranged into a target matrix $\mathbf{Y}_{target} \in \mathbb{R}^{N_Y \times P}$. For tree-to-element transductions, input patterns in $\mathscr{T}$ correspond to trees and the columns of matrix $\mathbf{X}$ contain the states obtained by applying the state mapping function to the corresponding structured states computed by the reservoir. Matrix $\mathbf{W}_{out}$ is therefore selected to solve the least squares linear regression problem:

$$\min \|\mathbf{W}_{out}\mathbf{X} - \mathbf{Y}_{target}\|_2^2 \tag{17}$$

In this paper, to solve equation 17 we use both Moore-Penrose pseudo-inversion of matrix $\mathbf{X}$ and ridge regression. In the first case, the reservoir-to-readout matrix is computed as

$$\mathbf{W}_{out} = \mathbf{Y}_{target}\mathbf{X}^{+} \tag{18}$$

where $\mathbf{X}^+$ denotes the pseudo-inverse of $\mathbf{X}$. In the second case, $\mathbf{W}_{out}$ is computed as

$$\mathbf{W}_{out} = \mathbf{Y}_{target}\mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda_r \mathbf{I}_{N_R})^{-1} \tag{19}$$

where $\mathbf{I}_{N_R}$ is the identity matrix of size N_R and λ_r is the regularization parameter (which determines the magnitude of the readout weights).

3.4. Computational Complexity of TreeESNs

In this section we provide an analysis of the computational complexity of the TreeESN model. For each input tree $\mathbf{t}$, the encoding process consists in the computation of the state for each node in $N(\mathbf{t})$ using equation 12. It is straightforward to see that the application of equation 12 to a node n requires a number of $O(kRN_R)$ operations, where k is the maximum degree over the set of trees considered, R is the maximum number of connections for each reservoir unit (R is smaller for sparser reservoirs) and N_R is the dimension of the reservoir. The total cost of the encoding process on tree $\mathbf{t}$ is

$$O(|N(\mathbf{t})|kRN_R) \tag{20}$$

which scales linearly with both the number of nodes in the input tree and the number of units in the reservoir. Note that the cost of the encoding in TreeESNs is the same for training and testing, resulting in a very efficient approach. For the sake of comparison, training the recurrent connections in standard RecNNs, even using efficient learning algorithms such as the Back-propagation Through Structure [21, 48], requires the extra cost (both in memory and in time) for the gradient computations for each training epoch. Analogously, extra-costs for gradient computations for recurrent connections are required also by RelNNs and GNNs. In particular, GNN is based on a iterative training algorithm, which alternates state relaxation and gradient computation phases, possibly requiring thousands of learning iterations [32]. The TreeESN model compares well also with state-of-the-art kernel methods for tree structured data. Here we limit our discussion to kernels considered in this paper for performance comparison, and make the assumption of a bounded maximum degree k. The cost of the encoding (considered with respect to the number of nodes) scales quadratically for the Partial Tree (PT) [37], Subset Tree (SST) [39], Route [40], q-gram and q-gram based kernels [49], Yamanishi [50], Subtree (ST) [38] and Subpath [51] kernels. As a particular case, the ST and the Subpath kernels can be computed in linear and log-linear (on average) time, respectively, under the assumption of a finite set of input labels. In the cases of the Activation Mask (AM) [52] and of the AMπ [53] kernels, the encoding includes a stage involving a preliminary training of a Self-Organizing Map for structured data (SOM-SD) [46], possibly requiring hundreds of learning iterations.

As regards the state mapping function for TreeESNs implementing tree-to-element transdutions, note that its cost is constant for the root state mapping and linear in the number of nodes and reservoir units for the mean state mapping.

The cost of training the linear readout in a TreeESN depends on the method used to solve the linear least squares problem of equation 17. This can range from a direct method e.g. using Moore-Penrose pseudo-inversion computed by singular value decomposition, whose cost is cubic in the number of training patterns (trees in our cases), to efficient iterative approaches for which the cost of each epoch is linear in the number of training patterns. Note that the cost of training the extremely simple readout tool in TreeESNs, i.e. a single layer of feed-forward linear units, is in general inferior to the cost of training more complex readout implementations, such as Multi-layer Perceptrons (often used in RecNN and related approaches) or SVMs, used in kernel methods.

3.5. Markovian Characterization and Initialization of Reservoir Dynamics

In the context of sequence processing it is a known fact that state models implementing contraction mappings are characterized by a Markovian nature of the state dynamics. Relations between contractions and Markovianity have been investigated in the contexts of Iterated Function Systems (IFS), variable memory length predictive models, fractal theory and for describing the bias of trainable RNNs initialized with small weights [17, 54, 18, 19]. In fact, it has been shown in [17, 18] that RNNs initialized to implement *contractive* state transition functions are *architecturally biased* towards Markovian process modeling, being able to discriminate among different input histories in a *suffix-based* Markovian flavor even prior to training of the recurrent connections. ESNs, through a fixed contractive setting of the state transition function, directly exploit the Markovian nature of state dynamics for efficiently approaching tasks within the architectural bias of RNNs, without any adaptation of the recurrent state transition function parameters [19, 20].

Preliminary results on a similar architectural bias towards Markovian models on tree domains for RecNNs initialized with contractive recursive state transition functions have been presented in [34]. In particular, in [34] it has been proved that the class of RecNNs with contractive state transition function and bounded state space is equivalent to (can be approximated arbitrarily well by) the class of models on tree domains with state space organization of Markovian nature. Therefore, TreeESNs, implementing fixed contractive recursive state transition functions, allow to extend the advantages of the RC approach to RecNN modeling.

3.5.1. Markovianity of TreeESN dynamics

For the aims of this paper, we say that a state model on tree domains is characterized by a state space organization of a *Markovian nature* whenever the states it assumes in correspondence of different input trees *sharing a common suffix* are close to each other proportionally to the height of the common suffix [34].

In TreeESNs, the contractive setting of the state transition function implies a Markovian characterization of the reservoir state space. For our purposes, the definition of contractivity of the node-wise encoding function τ (equation 5) implemented by the recursive state transition function of the reservoir (equation 12) is given as follows. The node-wise encoding function $\tau : \mathbb{R}^{N_U} \times \mathbb{R}^{kN_R} \to \mathbb{R}^{N_R}$ is a *contraction* with respect to the state space $\mathbb{R}^{N_R}$ if there exists a non-negative parameter $C < 1$ such that for every $\mathbf{u} \in \mathbb{R}^{N_U}$ and for every $\mathbf{x}_1, \ldots, \mathbf{x}_k, \mathbf{x}'_1, \ldots, \mathbf{x}'_k \in \mathbb{R}^{N_R}$ it holds that

$$\|\tau(\mathbf{u}, \mathbf{x}_1, \ldots, \mathbf{x}_k) - \tau(\mathbf{u}, \mathbf{x}'_1, \ldots, \mathbf{x}'_k)\| \leq C \max_{i=1,\ldots,k} \|\mathbf{x}_i - \mathbf{x}'_i\| \qquad (21)$$

Whenever the state transition function of a TreeESN is contractive according to equation 21 and the network state space is bounded, the nature of the reservoir dynamics is characterized by Markovianity. Intuitively, the roots of different input trees sharing a common suffix are mapped into reservoir states which are close to each other proportionally to the height of the common suffix. More formally, the Markovian property of a TreeESN reservoir space can be described in the following way. Consider a TreeESN with recursive state transition function τ implementing a contraction with parameter C with respect to the state space $\mathbb{R}^{N_R}$. Suppose that the subset of states which are assumed by the reservoir of the TreeESN is bounded with diameter denoted by *diam*. Then, for every height $h > 0$, any two input trees $\mathbf{t}, \mathbf{t}' \in (\mathbb{R}^{N_U})^{\#}$ sharing the same suffix of height h, i.e. $S_h(\mathbf{t}) = S_h(\mathbf{t}')$, and any states $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^{N_R}$, the distance between the states computed by the reservoir for the root of the two input trees $\mathbf{t}$ and $\mathbf{t}'$, with null states $\mathbf{x}$ and

$\mathbf{x}'$ respectively, is upper bounded by a term proportional to C^h:

$$\|\hat{\tau}(\mathbf{t},\mathbf{x}) - \hat{\tau}(\mathbf{t}',\mathbf{x}')\| \leq C^h \; diam \tag{22}$$

In fact:

$$\begin{aligned}
&\|\hat{\tau}(\mathbf{t},\mathbf{x}) - \hat{\tau}(\mathbf{t}',\mathbf{x}')\| = \\
&\|\tau(\mathbf{u}(root(\mathbf{t})), \hat{\tau}(\mathbf{t}(ch_1(root(\mathbf{t}))),\mathbf{x}), \ldots, \hat{\tau}(\mathbf{t}(ch_k(root(\mathbf{t}))),\mathbf{x})) - \\
&\quad \tau(\mathbf{u}(root(\mathbf{t})), \hat{\tau}(\mathbf{t}(ch_1(root(\mathbf{t}'))),\mathbf{x}'), \ldots, \hat{\tau}(\mathbf{t}(ch_k(root(\mathbf{t}'))),\mathbf{x}'))\| \leq \\
&C \max_{i_1=1,\ldots,k} \|\hat{\tau}(\mathbf{t}(ch_{i_1}(root(\mathbf{t}))),\mathbf{x}) - \hat{\tau}(\mathbf{t}(ch_{i_1}(root(\mathbf{t}'))),\mathbf{x}'\| \leq \\
&C^2 \max_{i_1,i_2=1,\ldots,k} \|\hat{\tau}(\mathbf{t}(ch_{i_2}(ch_{i_1}(root(\mathbf{t})))),\mathbf{x}) - \hat{\tau}(\mathbf{t}(ch_{i_2}(ch_{i_1}(root(\mathbf{t}')))),\mathbf{x}'\| \leq \\
&C^h \max_{i_1,\ldots,i_h=1,\ldots,k} \|\hat{\tau}(\mathbf{t}(ch_{i_h}(\ldots(ch_{i_1}(root(\mathbf{t})))\ldots)),\mathbf{x}) - \\
&\qquad\qquad \hat{\tau}(\mathbf{t}(ch_{i_h}(\ldots(ch_{i_1}(root(\mathbf{t}')))\ldots)),\mathbf{x}')\| \leq \\
&C^h diam
\end{aligned} \tag{23}$$

As a consequence of this Markovian property, the distance between the reservoir states computed for nodes whose induced sub-trees are different only before a common suffix of height h is anyhow bounded by a term which exponentially decreases for increasing h. Equivalently, the influence on the reservoir state computed for node n (i.e. $\mathbf{x}(n)$) due to nodes at depth d in the sub-tree rooted at n (i.e. $\mathbf{t}(n)$) is exponentially decreasing for increasing d. Hence, the reservoir of a TreeESN is able to discriminate between different input trees in a Markovian tree suffix-based way without any adaptation of its parameters. A straightforward consequence is that TreeESNs result in a very efficient RecNN approach particularly suitable for tasks in which the target characterization is compatible with such Markovian state space organization.

The Markovian nature of TreeESN reservoir state space organization is also graphically illustrated in Figure 8, which shows different input trees and corresponding states for their root nodes, computed by a model obeying Markovian organized dynamics. The states computed for the roots of the different input trees cluster together in a tree suffix-based fashion. In

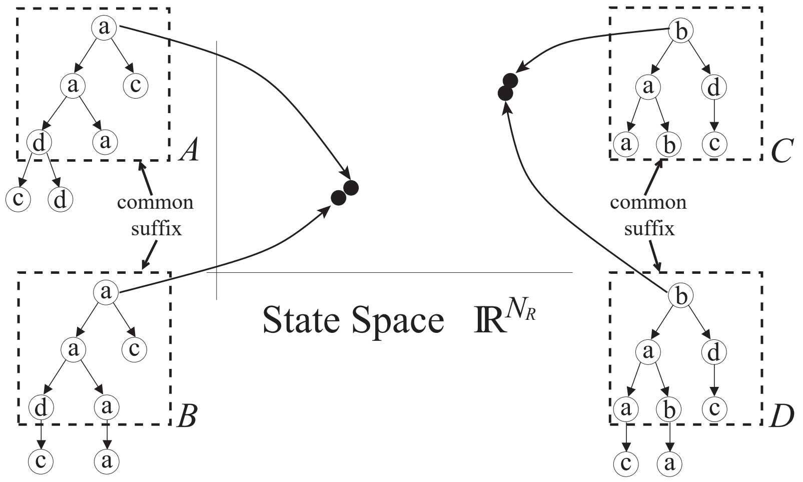


Figure 8: Graphical illustration of the Markovian nature of reservoir state space organization in TreeESNs. The root of input trees with shared suffix correspond to close points in the reservoir state space $\mathbb{R}^{N_R}$ (see equation 22).

particular, the roots of trees A and B are mapped into very close state representations with respect to the state representation of the root of C and D. Thereby, the readout of the network can naturally perform better whenever the target of the task at hand requires to assign similar outputs for the roots of A and B and a different one for the root of C and D. On the contrary, the Markovian constrained state space turns out to be less appropriate e.g. for targets requiring similar outputs for the roots of A and C and a different one for the root of B.

The example in Figure 8 corresponds to the case of tree-to-element transductions computed using root state mapping (and to the case of tree-to-tree transductions). However, note that in general, for tree-to-element transductions, the Markovian characterization of reservoir dynamics, holding locally in correspondence of each node of an input tree, can be mitigated as a global property of the feature representation used to feed the readout, according to the state mapping function adopted. This aspect is discussed in Section 3.5.3.

3.5.2. Reservoir Initialization

The recursive state transition function (equation 12) of a TreeESN is initialized to implement a contraction mapping (equation 21) with a bounded state space. Under such conditions, the reservoir state dynamics are stable and characterized by Markovianity as discussed in Section 3.5.1.

The condition on the bounded reservoir state space is ensured under very mild assumptions, e.g. for a bounded reservoir activation function such is *tanh*.

As regards the contractivity of the reservoir state transition function, we provide a condition assuming the Euclidean distance as metric on $\mathbb{R}^{N_R}$ and *tanh* as activation function of the reservoir units[3]. For every input label $\mathbf{u} \in \mathbb{R}^{N_U}$ and children states $\mathbf{x}_1, \dots, \mathbf{x}_k, \mathbf{x}'_1, \dots, \mathbf{x}'_k \in \mathbb{R}^{N_R}$:

$$\begin{aligned}
&\|\tau(\mathbf{u}, \mathbf{x}_1, \dots, \mathbf{x}_k) - \tau(\mathbf{u}, \mathbf{x}'_1, \dots, \mathbf{x}'_k)\|_2 = \\
&\|tanh(\mathbf{W}_{in}\mathbf{u} + \sum_{i=1}^{k} \hat{\mathbf{W}}\mathbf{x}_i) - \ \ tanh(\mathbf{W}_{in}\mathbf{u} + \sum_{i=1}^{k} \hat{\mathbf{W}}\mathbf{x}'_i)\|_2 \leq \\
&\|\hat{\mathbf{W}} \sum_{i=1}^{k} (\mathbf{x}_i - \mathbf{x}'_i)\|_2 \leq \\
&\|\hat{\mathbf{W}}\|_2 \sum_{i=1}^{k} \|\mathbf{x}_i - \mathbf{x}'_i\|_2 \leq \\
&k\|\hat{\mathbf{W}}\|_2 \max_{i=1,\dots,k} \|\mathbf{x}_i - \mathbf{x}'_i\|_2
\end{aligned} \tag{24}$$

Contractivity of the state transition function is hence guaranteed by the condition:

$$\sigma = k\|\hat{\mathbf{W}}\|_2 < 1 \tag{25}$$

where k is the maximum degree over the set of trees considered and σ is called the *contraction coefficient* of the TreeESN, governing the degree of contractivity of the reservoir dynamics. A straightforward initialization procedure for TreeESNs then consists in a random setting of both the input-to-reservoir weight matrix $\mathbf{W}_{in}$ and the recurrent reservoir weight matrix $\hat{\mathbf{W}}$, after which $\hat{\mathbf{W}}$ is scaled such that $\sigma < 1$ (equation 25) holds. Elements in $\mathbf{W}_{in}$ can be chosen according to a uniform distribution over the interval $[-scale_{in}, scale_{in}]$ as in standard ESNs, where $scale_{in}$ determines the size of the *input scaling*.

Equation 25 generalizes the sufficient condition (on the maximum singular value of the reservoir recurrent matrix) for the ESP in standard ESN

[3]In [34] a similar condition is provided assuming the maximum norm as metric on the state space.

processing [15], that corresponds to the case $k = 1$ in equation 25. As for the sufficient condition for the initialization of standard ESNs, equation 25 is quite restrictive. Actually, even though the recursive state transition function τ is not contractive in the Euclidean norm, it could still be a contraction in another norm and the argumentation in equations 22 and 23 would hold anyway. For this reason, in this paper we also consider values of the contraction coefficient σ slightly greater than 1.

3.5.3. Markovian Characterization and Tree-to-element Transductions

For TreeESNs implementing tree-to-element transductions, the influence on the output of the network due to the Markovian characterization of state dynamics is influenced by the state mapping function adopted.

When the root state mapping is used, the only state information considered for the computation of the output corresponding to an input tree is the state computed for its root. In fact, the strong assumption underlying the application of the root state mapping with TreeESNs is that the (fixed) dynamics of interest to be caught for properly tackle the task at hand can be centered in the root of the input structure. In this case, the relevant Markovian characterization of reservoir dynamics is centered only in the root node as well. Accordingly, the suitability of TreeESN-R is limited to those tasks whose target dynamics is compatible with this root-focused Markovian characterization.

In the case of mean state mapping, the Markovian characterization of reservoir dynamics is mitigated by the use of the mean operator. Indeed, the state information used to feed the readout depends to the same extent on the states computed for every node in the input tree. This choice for the state mapping function might therefore result in a less restricting model, in particular for applications to tree-to-element tasks whose target dynamics is not suitable for any specific node-focused Markovianity.

The effects and limitations of the two proposed state mapping functions are analyzed through experiments on two ad-hoc defined target functions on trees with explicit Markovian and anti-Markovian characterization in Section 4.1.1, and further investigated on a real-world task in Section 4.1.2.

4. Experiments

In this Section we present the application of the TreeESN model to tasks on domains of different nature related to tree-to-element transductions. The aim is to support with empirical evidences the different characteristics of the approach. First, in Section 4.1 we focus on the analysis of the characteristics and of the limitations of the TreeESN approach due to the relation between the state mapping function and the Markovianity of the target. To this aim we take into consideration artificial Markovian/anti-Markovian tasks on trees and a task from a Chemical domain, already treated as tree domains by RecNN and kernel approaches, for which we can have a tight control of the target and data meaning (specifically in terms of Markovianity) either by construction (artificial data) or by know previous characteristic/results (e.g. [55]). Due the analysis purpose of these experiments, we report explicitly the results of the model with respect to different hyper-parameters setting. The aim is to characterize the analyzed phenomena also into different condition setting, instead of trying to optimize such setting for performance improvements.

Then, in Section 4.2 we evaluate the potentiality of such efficient approach considering several benchmarks that allow us to compare with a large set of state-of-the-art related models, considering both the efficiency and the predictive performance (evaluated according to the error measure reported in the corresponding literature). In this case we select the main TreeESN hyper-parameters through a model selection procedure based on cross-validation (thought without adopting further optimization approaches).

In all the experiments, we assume a basic general architecture (e.g. with full stationary condition of equation 12, linear readout). TreeESN-R and TreeESN-M are obtained by applying the different state mapping functions to the structured state representations computed by the same reservoirs.

4.1. Experimental Analysis of Markovian Properties for TreeESN

In this Section we present and discuss the experimental analysis of TreeESN with respect to Markovianity and its relationship with root and mean state mapping functions.

4.1.1. Markovian/anti-Markovian Tasks

In order to have a tight control of the Markovian conditions, we introduce two new artificial regression tasks on tree structures with target functions characterized respectively by a distinct Markovian and anti-Markovian nature, designed to extend to tree domains the analogous tasks on sequences used in [41, 20].

The trees in the dataset have height between 3 and 15, and degree equal to 3. The skeleton of every tree was randomly generated in a top-down fashion, starting from the root and such that for every node the probability of an absent child is 0.4 for the first child and 0.7 for the other children. Symbolic node labels were assigned randomly using a uniform distribution over the alphabet $\mathcal{A} = \{a, b, \ldots, j\}$ and then mapped into the numerical set $\{0.1, 0.2, \ldots, 1.0\}$, such that a is represented by 0.1, b is represented by 0.2 and so on up to j represented by 1.0. The numerical label associated to node n is denoted by $u(n)$. Two examples of input trees in this dataset are reported in Figure 9. The number of trees in the dataset is 1000, of which 800 were used for training and 200 for testing. The number of nodes in the trees is highly variable between 4 and 478.

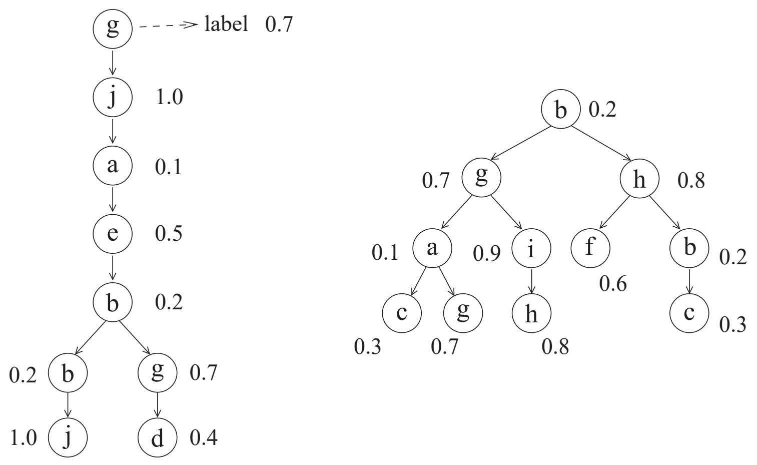


Figure 9: Two examples of input trees in the Markovian/anti-Markovian dataset.

The two different tasks were obtained by associating different target outputs with Markovian or anti-Markovian flavor to the same trees in the dataset, using a parameter $\lambda > 1$ to control the degree of Markovianity/anti-Markovianity of the task.

For the Markovian task, the target associated to each tree $\mathbf{t}$ was computed as follows:

$$y_{target}(\mathbf{t}) = \sum_{n \in N(\mathbf{t})} \frac{u(n)}{\lambda^{depth(n,\mathbf{t})}} \tag{26}$$

such that the contribution of each node n to the target value for $\mathbf{t}$ exponentially decreases with the depth of n.

For the anti-Markovian task, the target function is defined as:

$$y_{target}(\mathbf{t}) = \sum_{n \in N(\mathbf{t})} \frac{u(n)}{\lambda^{(h(\mathbf{t}) - depth(n,\mathbf{t}))}} \tag{27}$$

in which case the contribution of node n to the target $y_{target}(\mathbf{t})$ exponentially increases with the depth of n.

The target values computed for both the tasks, according to equations 26 or 27, respectively, were normalized in $[-1, 1]$. In our experiments, we used the value of $\lambda = 2$ for both the tasks.

On the Markovian/anti-Markovian tasks, we tested TreeESNs with 200-dimensional sparse reservoirs with 40% of connectivity, input scaling $scale_{in} = 1$ and contraction coefficient $\sigma \in \{0.5, 1, 1.5, 2, 2.5\}$. For each value of σ, the results are averaged over 30 independently generated random guessed reservoirs. The readout was trained by using pseudo-inversion (eq. 18).

Figures 10 and 11 show the Mean Absolute Error (MAE) and the standard deviation (among the 30 guesses) on the Markovian and anti-Markovian tasks, respectively, obtained by TreeESNs in correspondence of both the choices for the state mapping function. For the sake of comparison, in the same figures we also report the error obtained by the *null model* whose output is always equal to the target output value averaged on the training set.

The errors of TreeESNs on both the tasks are only slightly influenced by the value of σ, while the choice of the state mapping function reveals a deep impact on the performance. Indeed TreeESN-R outperforms TreeESN-M on the Markovian task for every value of σ, while on the anti-Markovian task the performance of TreeESN-M is always better than that of TreeESN-R, which in turn is worse than the result obtained by the null model. These results enlighten the effects of the different organizations of the feature spaces corresponding to the different state mapping functions considered. In particular, TreeESN-R has a good performance on the Markovian task, whose target is designed to match the nature of reservoir dynamics, with a specific reference to the root-focused Markovianity (see Section 3.5), whereas a poorer performance than the null model is obtained on the anti-Markovian task, whose target has an opposite characterization. On the other hand, the use of the mean state mapping function has the effect of merging the states computed for all the nodes in the input tree such that the relevance of suffixes and prefixes on the feature state that feeds the readout is the same. Accordingly, the characterization of the resulting TreeESN-M model can be considered

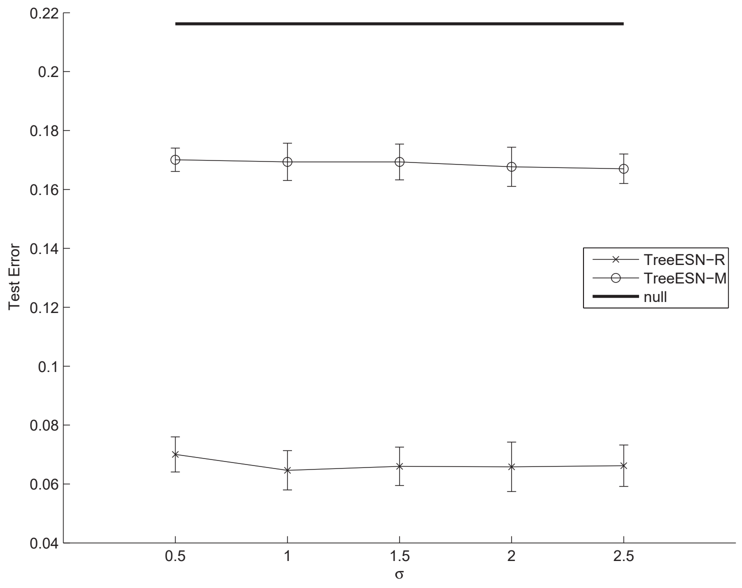


Figure 10: MAEs and standard deviations on the test set of the *Markovian* task for TreeESN-R, TreeESN-M and a null model.

as the middle between Markovianity and anti-Markovianity. Compared to the case of TreeESN-R, the feature space in TreeESN-M is less suitable for the Markovian task and worse results are obtained (thought it improves the null model). TreeESN-M clearly leads to better results than TreeESN-R on the anti-Markovian task. Nevertheless, we note that resorting to the mean operator is not sufficient by itself to appropriately overcome the unsuitability of reservoir dynamics for the task. This can be observed by comparing the scale of the performance of TreeESN-R on the Markovian task (Figure 10) with the scale of the performance of TreeESN-M on the anti-Markovian one (Figure 11). However, it is worth noticing that TreeESN-M achieves rather better performances on the anti-Markovian task than on the Markovian one. This is interestingly related to the nature of the concept of anti-Markovianity on tree domains. On tree domains prefixes and suffixes are not symmetric as for sequences. Indeed for anti-Markovian target functions on trees, the relevance of each node on the target output exponentially increases with the depth of the node. As the number of high-depth nodes (constituting the prefixes) increases by construction with the depth, the average of the states

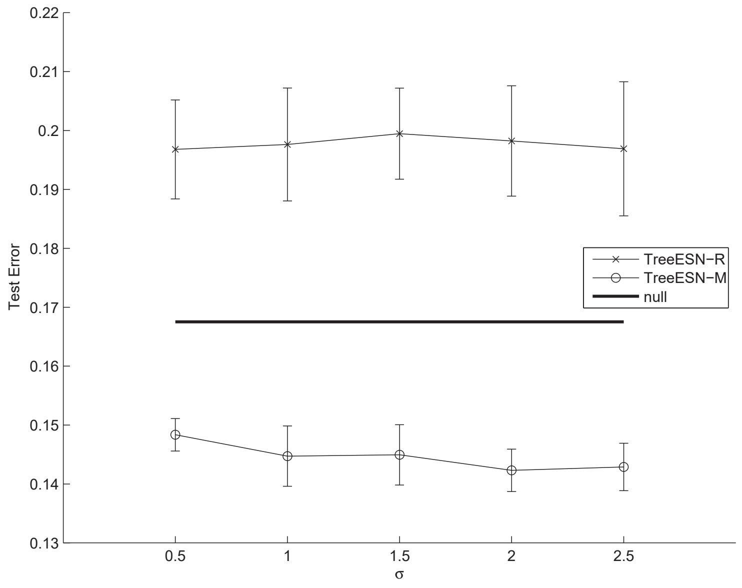


Figure 11: MAEs and standard deviations on the test set of the *anti-Markovian* task for TreeESN-R, TreeESN-M and a null model.

over all the nodes in the input tree is more strongly influenced by the states of higher depth nodes, further enhancing the suitability of TreeESN-M for tree anti-Markovian tasks.

4.1.2. Alkanes Task

Alkanes task allow us to show the possible effects of the Markovian condition on a real-world task. This is a regression task related to a *Quantitative Structure Property Relationship* (QSPR) analysis of alkanes [25, 55], consisting in predicting the boiling point (measured in Celsius degrees, ^{o}C) of such molecules on the basis of their tree structure representation.

The dataset consists in 150 alkanes. The dataset is fully reported in [25] and is also available at http://www.di.unipi.it/∼micheli/dataset, with target boiling temperatures varying from $-164\ ^{o}C$ to $174\ ^{o}C$. As introduced in [25], every molecule is represented by a tree (rooted according to chemical rules) in which nodes stand for atoms and edges between nodes stand for bonds. In particular, since only carbon atoms are considered in the molecular descriptions (i.e. hydrogens are suppressed), the node labels used are 1-dimensional and all equal to 1.0. Figure 12 shows an example of tree representation for an alkane molecule. The degree of the set of trees considered in this dataset is $k = 3$ and the maximum number of nodes in a tree is 10.

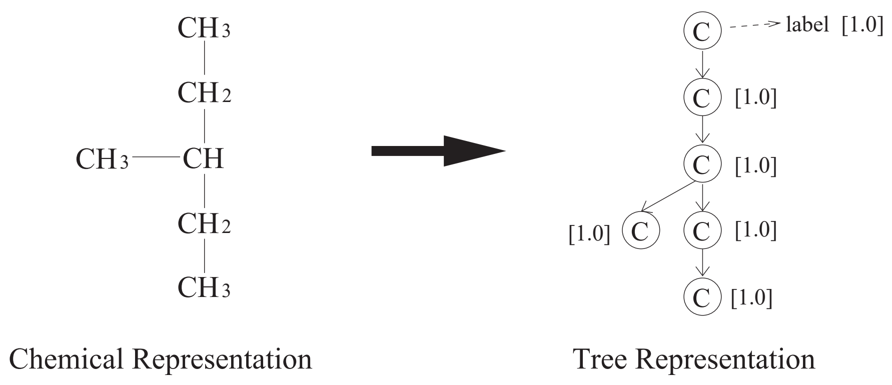


Figure 12: Tree representation (hydrogens are suppressed) of the 3-methylpentane molecule in the Alkanes dataset.

For our analysis aim alkanes represent a class of molecules with rather uniform and systematic structure, including all the possible configurations up to 10 carbon atoms. Moreover, the target boiling point temperature is known to be related to global molecular properties such as the number of atoms of the main chain (molecular size) and the pattern of atom branching (molecular shape). The fact that such global characteristics do not depend on the suffixes of the alkanes trees suggest us that the target does not agree to Markovian assumptions, as we will detail in the following, making this task particularly useful to distinguish the role of TreeESN-R TreeESN-M models in a real-world task.

RecNNs and other state-of-the-art learning models for tree domains have been applied to this dataset. Thought the main aim in this section is for analysis purpose, this task gives us also the opportunity to have a a first rough performance comparisons with other models. More specifically, we take into consideration the application to this task of Recursive Cascade Correlation (RCC) [25], Contextual Recursive Cascade Correlation (CRCC) [45], a variant of the SST kernel [56] and Neural Networks for Graphs (NN4G) [55].

On the Alkanes dataset we tested TreeESNs with reservoir dimension varying between 10 and 70 (with a step of 5) and with 40% of connectivity among reservoir units. Input scaling was set to $scale_{in} = 0.5$ and the values of the contraction coefficient considered were $\sigma \in \{1, 2\}$. For every parametrization of the reservoir, we averaged the results over 30 independently generated reservoirs. The readout was trained by pseudo-inversion (eq. 18). Results are fully reported for each parametrization and not used for a model selection (as explained at the beginning of Section 4). Test results presented here were obtained by using a 10-fold cross validation procedure.

Figures 13 and 14 show the MAE and standard deviation (averaged over the 10 folds) on the test set obtained by TreeESN-R and TreeESN-M in correspondence of the two different value of the contraction coefficient σ. It is evident that the performance of the model is sensible to the choice of the state mapping function. TreeESN-M outperforms TreeESN-R for every reservoir dimension and value of σ, with a smaller performance variance as well. The best result achieved by TreeESN-M, in correspondence of $N_R = 40$ and $\sigma = 2$, is 2.78 ^oC. The poorer results of TreeESN-R provide an empirical evidence on the possible effect of non-Markovian characterization of the target.

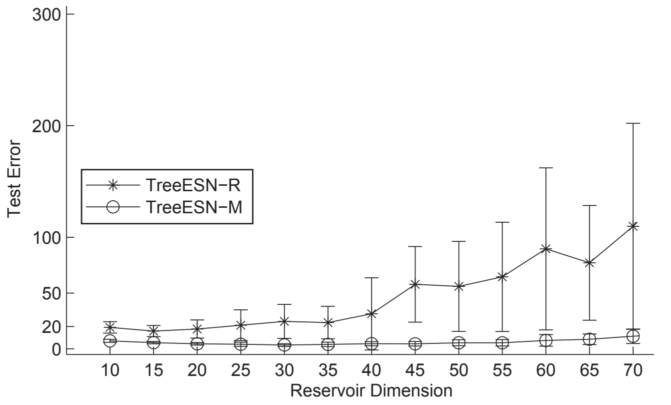


Figure 13: MAE and standard deviatio on the test set of the Alkanes dataset for TreeESN-R and TreeESN-M with $\sigma = 1$.

We also observed the influence of the value of the contraction coefficient σ ($\sigma = 2$, Figure 14, leads to better performances than $\sigma = 1$, Figure 13) and the reservoir dimensionality (with underfitting and overfitting behaviors observed for too small and too large reservoirs, respectively) on the absolute performance. However, it results that the qualitative effects of Markovianity and the relative differences on TreeESN-R and TreeESN-M results are not due to the effect of the choice of such parameters values.

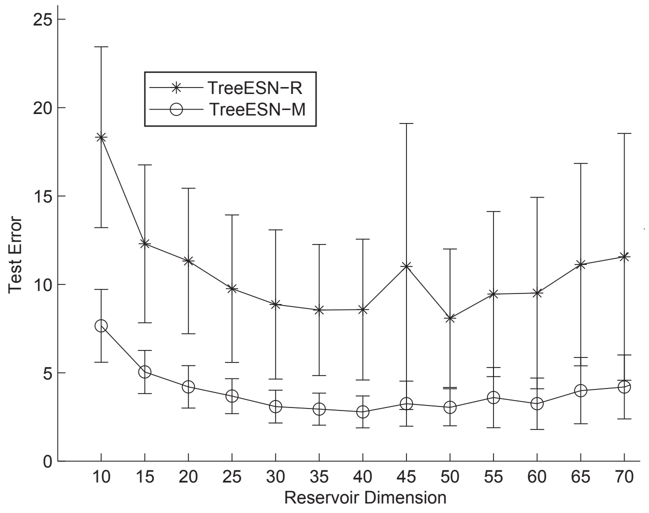


Figure 14: MAE and standard deviation on the test set of the Alkanes dataset for TreeESN-R and TreeESN-M with $\sigma = 2$.

The gap between TreeESN-R and TreeESN-M can be further observed in Table 1, showing that only TreeESN-M achieves reasonable results with respect to the state-of-the-art. In fact, although the experiments on the Alkanes dataset were conceived for analysis purposes only, we anyway also compare in Table 1 the results otained by TreeESNs with those obtained by RCC, CRCC, SST kernel and NN4G under similar fitting conditions on the training set. In particular, in analogy to [45], we report the test errors in correspondence of $\sigma = 2$ for the smallest reservoir dimension yielding a maximum absolute error on the training set below the threshold $\epsilon_t = 8\ ^oC$. For the sake of comparison with NN4G [55], we also provide the results corresponding to $\epsilon_t = 5^oC$. In addition, to show the possible range of performances on this task, the best TreeESNs results (corresponding to the minimum MAE on the test set) are reported in Table 1 as well.

Note that the performance of TreeESN-M, although obtained by an extremely efficient model with fixed causal encoding and linear readout, is in line with those achieved by more complex learning models for structured data. In addition, the largest test errors of TreeESNs are observed in correspondence of the smallest alkanes, which are in a high non-linear relation with their target boiling point.

In order to clarify the role of the Markovianity in the reported results, it

Model	ϵ_t	**Test Set MAE**
TreeESN-R	*best*	8.09(±3.91)
TreeESN-R	8^oC	15.01(±9.24)
TreeESN-R	5^oC	13.18(±8.58)
TreeESN-M	*best*	2.78(±0.90)
TreeESN-M	8^oC	3.09(±0.93)
TreeESN-M	5^oC	3.05(±1.05)
RCC	8^oC	2.87(±0.91)
CRCC	8^oC	2.56(±0.80)
SST	8^oC	2.93(±0.92)
NN4G	8^oC	2.34(±0.31)
NN4G	5^oC	1.74(±0.23)

Table 1: MAE on the test set (expressed in ^oC degrees) and corresponding standard deviation for TreeESNs ($\sigma = 2$) and other learning models for tree domains on the Alkanes dataset.

is interesting to directly analyze the organization of the feature spaces arising from the application of the different state mapping functions in TreeESNs. To this aim we refer in particular three examples of alkane molecules in Figure 15, whose respective tree representations share the same suffix of height 3, but are associated to very different values of the target boiling point, corresponding to different molecular size and shape. Such cases well represent a frequent occurrence of the known characteristic of this dataset, as mentioned above.

For visualization aim, we applied Principal Component Analysis (PCA) to the feature representations of the alkanes computed by TreeESNs. Figure 16 shows the plot of the first two principal components of the feature space for a TreeESN-R and for a TreeESN-M in correspondence of $N_R = 50$ and $\sigma = 2$.

According to the Markovian organization of the reservoir dynamics preserved by the the root state mapping function (TreeESN-R), Figure 16(a) clearly shows that the feature states computed for the molecules in the dataset are clustered together according to the suffix of the input structures (represented in the same figure under the corresponding cluster). In particular, the trees in Figure 15 sharing a long common suffix are mapped into very close states (see the A, B, C labels). Since the target values of A, B, C molecules are very different, this Markovian organization is very unsuitable for the task.

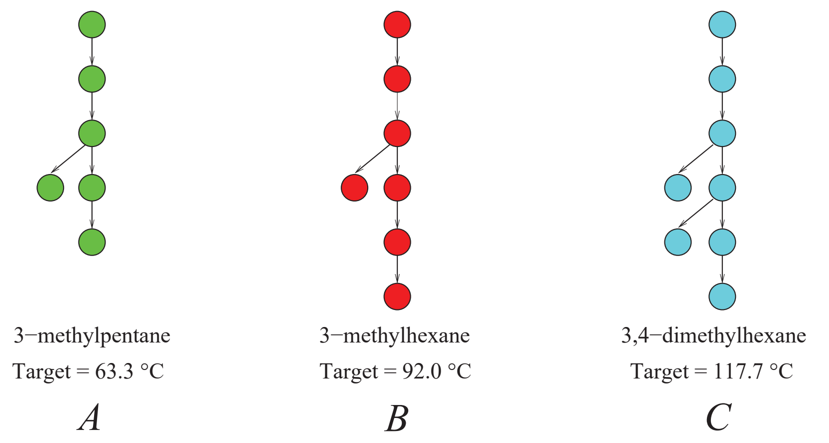
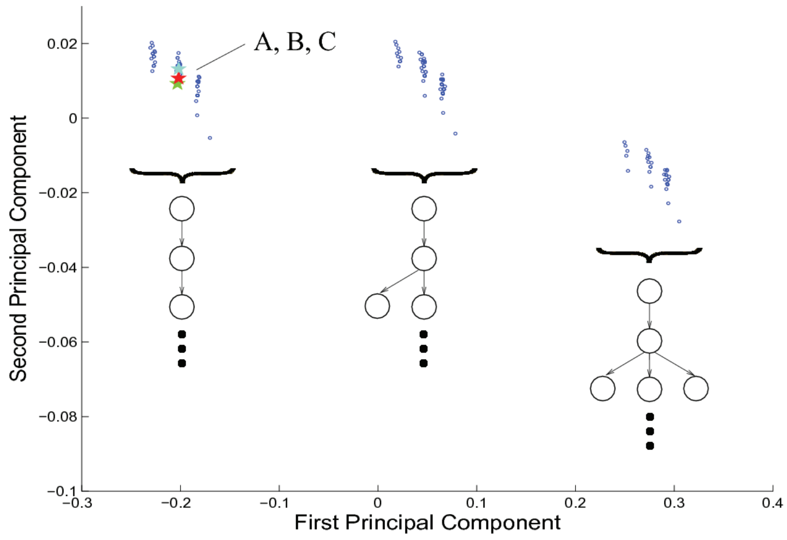
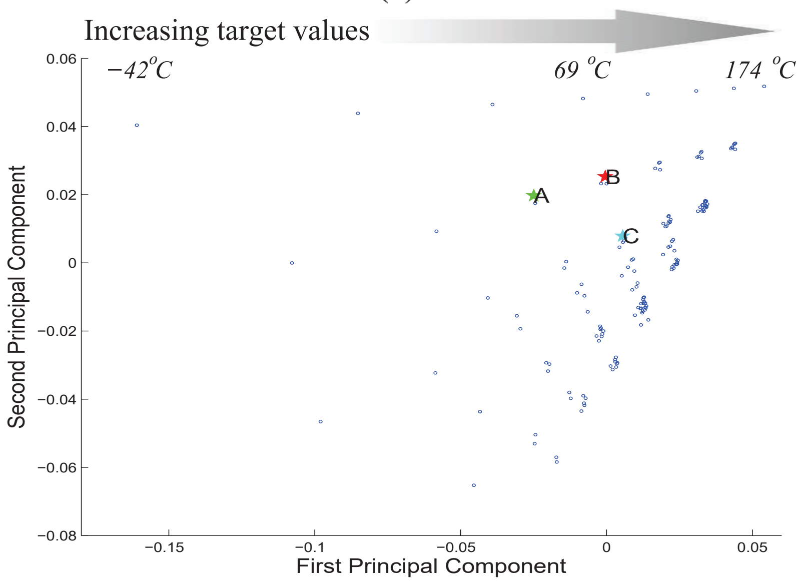


Figure 15: Examples of tree representations of molecules in the Alkanes dataset with common suffix of height 3 and different target values. Molecule A has target 63.3 ^oC, molecule B has target 92.0 ^oC and molecule C has target 117.7 ^oC.

The use of the mean state mapping function (TreeESN-M) influences the feature space organization such that Markovianity of reservoir dynamics is no longer preserved. In this case, the feature space results organized according to the size and shape of the input structures. In Figure 16(b), the first principal component distributes the feature representations according to the number of carbon atoms in the main chain of the corresponding molecule, whereas the second principal component distributes them on the basis of the pattern of atom branching. Considering the trees in Figure 15, we can indeed observe that the states corresponding to molecules B and C, having the same number of atoms in the main chain, are mapped into close values of the first principal component. Analogously, the states for molecules A and B, with the same pattern of branching, are mapped into close values of the second principal component. Thus, this organization matches the known characteristics of the target. In particular, molecules such are A, B, C with very different target label result in well distinguished positions in the plot, which is more suitable for the task with respect to the case in Figure 16(a). For the general case, the arrow in Figure 16(b) approximatively shows the direction of increasing target values for all the molecules in the plot, that agrees with the distribution of their corresponding feature representations. Hence, the organization arising with TreeESN-M fits better the characteristics of the target, resulting in a facilitation for the linear readout tool and



(a)



(b)

Figure 16: Plots of the first two principal components of the reservoir state space computed by TreeESNs for the Alkanes dataset. The labels *A*, *B* and *C* refer to the molecules in Figure 15. Plot **(a)**: TreeESN-R, the shared suffixes of height 2 of the molecules are represented below each cluster. Plot **(b)**: TreeESN-M, the arrow on the top shows the distribution of the increasing target values.

then in better predictive performances.

4.2. Performance Evaluation of TreeESN

In this Section we present and discuss the potentiality of the TreeESN approach, considering both predictive performance and computational cost in practical applications in comparison with state-or-the-art neural and kernel-based models. In particular, we considered 8 predictive tasks of different nature, namely 5 tasks related to the prediction of aggregation functions, a multi-classification task in the field of document processing derived from the INEX2006 international competition, and 2 tasks from glycobiology related to classification of glycan molecules.

For model selection, in the following experiments, we considered different hyper-parameterizations for reservoir (including the reservoir dimension, the scaling of the matrices collecting input-to-reservoir and internal recurrent reservoir connections weights) and for readout regularization, which have been shown to be relevant for the RC models performance (e.g. [57, 13, 58, 59]).

As basic experimental setup for our experiments on TreeESNs, we used reservoirs with 40% of connectivity, contraction coefficient $\sigma \in \{1, 2, 3\}$ and input scaling parameter $scale_{in} \in \{0.1, 1\}$. We explored reservoir configurations varying the number of reservoir units in the range 10-1000, with specific settings of the range depending on the characteristics of the different datasets and tasks (as detailed in the following sub-sections). For every setting of the reservoir hyper-parametrization we independently generated a number of 10 random guessed reservoirs of TreeESNs and averaged the results over the guesses. Readouts were trained using pseudo-inversion and ridge-regression with regularization parameter $\lambda_r \in \{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}\}$. Model selection and performance assessment are based on cross-validation techniques. The details of the splitting in training, validation and test sets, and of the model selection procedure adopted, depend on the characteristics of the available datasets and are described in the following sub-sections. Note that distinct model selection procedures were applied for the two possible choices of the state mapping function, i.e. for TreeESN-R and TreeESN-M.

The efficiency of TreeESNs, already discussed theoretically in Section 3.4, is concretely shown by reporting the times required for training and testing the model on the different tasks. In practice, such timings were found to be almost the same for TreeESNs using root and mean state mappings, hence the reported values were averaged over the choice of the state mapping.

4.2.1. Aggregation Functions

The aggregation function experiments allow us to have a direct comparison with respect to the class of recursive neural networks models, for which TreeESN represent an architectural baseline. Such set of benchmarks have been used to asses the capability of GNN and RelNN models in grasping basic relational dependencies on tree structured data.

An aggregation function is a function applied to a bag of tuples (i.e. a set of multi-dimensional values), returning an output which is related to a particular property of the tuples in the bag. We considered 5 artificial aggregation functions datasets, constructed analogously to [31], in correspondence of the following 5 aggregation functions: count, sum, maximum, average and median. Each tuple contained 5 real values from a symmetric interval, where each bag contained a number of tuples between 5 and 10. Each aggregation function allowed for the definition of a regression task, such that for each bag the target is defined as the output of the aggregation function applied to the first field of each tuple in the bag. For instance, for the max function, the target associated to a bag of tuples is the maximum value among the first fields in each tuple of the bag. An exception to this is represented by the count function, in which case the target for each bag is assigned to the number of tuples in the bag. The artificial datasets were obtained by sampling targets values for the aggregation functions and then generating the values in the tuples in an appropriate range as in [31]. Bags of tuples were represented as rooted trees, where the leaf nodes stand for the tuples and the root stands for the bag. Zero labels were assigned to root nodes, while a 5-dimensional real label was associated to each leaf node, containing the values in the corresponding tuple (see Figure 17). In line with [31], each dataset contained 500 trees, split in training, validation and test sets with 300, 100 and 100 trees, respectively.

We used TreeESNs with $N_R \in \{10, 50, 100\}$ units[4], while all other experimental conditions were set as described in Section 4.2. As in [31], each bag was considered correctly predicted if the absolute difference between the target and the output of the TreeESN was not greater than 0.1. In the case of the count function, the output of the TreeESN was rounded to the closest

[4]Preliminary experiments on these datasets pointed out that a parameter search for model selection, considering reservoirs with dimension in the range 10-100, was sufficient for the characteristics of the tasks.

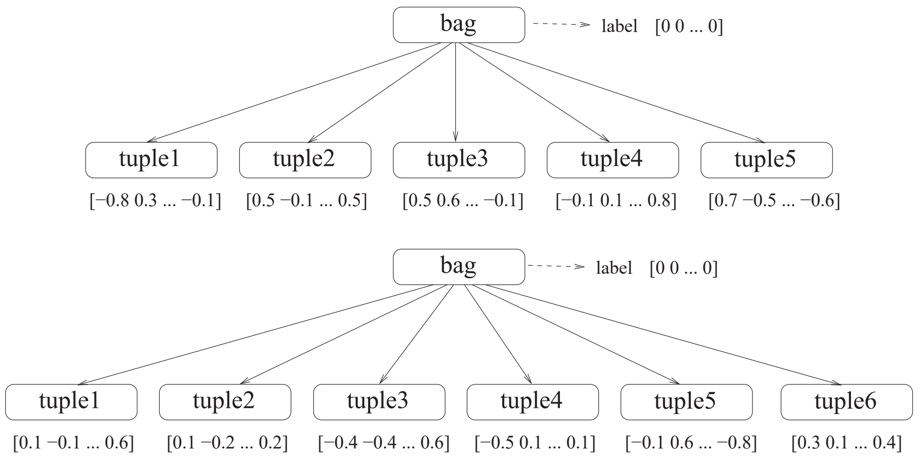


Figure 17: Two examples of tree representations of bag of tuples in the aggregation functions datasets.

integer. Reservoir hyper-parametrization and readout regularization were chosen on the validation set for each of the 5 tasks.

Results achieved by TreeESNs, in correspondence of root and mean state mappings, are reported in Tables 2, 3, 4, 5 and 6 for the count, sum, max, average and median aggregation functions tasks, respectively. In the same tables, the performance obtained by GNNs and RelNNs is presented as well. In particular, the performance of GNN is averaged over different representations of the input structures, while the performance of RelNN is averaged over possible architectural variants of the model (see [31] for details). In addition, for a further baseline comparison, the performance of the null model is reported as well.

In general, despite the fact that TreeESNs represent an architectural baseline for the more complex neural networks models with trained recursive dynamics considered, TreeESNs results on the aggregation functions tasks are comparable with those obtained by GNNs and RelNNs. Such comparison is particularly favorable, for instance, for the max and for the median aggregation functions (see Tables 4 and 6).

The results achieved by TreeESN-R and TreeESN-M on the aggregation functions tasks can further point out some interesting characteristics of the model in relation to the choice of the state mapping function. For instance, for the count task (see Table 2), the state computed for the root node of each

Model		Test Accuracy (%)
TreeESN-R		100.0
TreeESN-M		94.9
GNN	*mean*	83.5
	min-max	17.7-100.0
RelNN	*mean*	99.3
	min-max	99.0-99.9
null		17.0

Table 2: Average test accuracy (in %) on the *count* aggregation function task for TreeESNs, GNNs and RelNNs.

Model		Test Accuracy (%)
TreeESN-R		100.0
TreeESN-M		98.5
GNN	*mean*	98.1
	min-max	90.7-100.0
RelNN	*mean*	99.8
	min-max	99.7-99.8
null		16.0

Table 3: Average test accuracy (in %) on the *sum* aggregation function task for TreeESNs, GNNs and RelNNs.

tree is strongly related to the number of leaf nodes in the tree. Accordingly, TreeESNs with root state mapping are able to learn this task with 100% of test accuracy. On the other hand, the use of a mean state mapping has the effect of mixing together the state information computed in correspondence of the different nodes in the tree, reducing the influence of the state of the root on the final output and resulting in worse accuracy results.

The efficiency of the TreeESN approach is shown by reporting in Table 7 a comparison on training and test timings for TreeESN, GNN and RelNN [31], averaged over the different aggregation functions tasks. Although experiments on GNNs and RelNNs were conducted using PCs with more powerful processors (see [31] for details), Table 7 clearly shows the computational advantage of the TreeESN approach. Indeed, while test times were almost the same, training a TreeESN required only ≈ 0.3 seconds, which is 2 and 3 orders of magnitude faster than training a RelNN and a GNN, respectively, on the same tasks. This evident advantage is due to the fact that in

Model		Test Accuracy (%)
TreeESN-R		84.6
TreeESN-M		65.1
GNN	*mean*	57.2
	min-max	48.3-71.0
RelNN	*mean*	60.3
	min-max	45.1-78.1
null		11.0

Table 4: Average test accuracy (in %) on the *max* aggregation function task for TreeESNs, GNNs and RelNNs.

Model		Test Accuracy (%)
TreeESN-R		100.0
TreeESN-M		100.0
GNN	*mean*	99.3
	min-max	97.0-100.0
RelNN	*mean*	98.3
	min-max	96.0-99.8
null		19.0

Table 5: Average test accuracy (in %) on the *average* aggregation function task for TreeESNs, GNNs and RelNNs.

TreeESNs only the feed-forward linear readout part is trained, while in GNNs and RelNNs also the weight values on the recurrent connections undergo an iterative training procedure.

4.2.2. INEX2006 Task

INEX2006 is a real-world challenging multi-classification task coming from the INEX 2006 international competition [60]. The dataset considered [52, 40] is derived from the IEEE *structure only* corpus describing the hierarchical tree structure of XML formatted documents from 18 different journals[5]. The journal corresponding to each document is used as target

[5]The INEX 2006 competition [60] comprised also other document classification tasks: a task based a richer version of the dataset considered in this paper, i.e. the IEEE *structure and content* corpus (see [61, 5]), containing also the textual content for each document, and two classification tasks based respectively on Wikipedia structure only and Wikipedia

Model		Test Accuracy (%)
TreeESN-R		100.0
TreeESN-M		100.0
GNN	*mean*	82.5
	min-max	78.3-87.0
RelNN	*mean*	81.0
	min-max	75.7-86.4
null		31.0

Table 6: Average test accuracy (in %) on the *median* aggregation function task for TreeESNs, GNNs and RelNNs.

Model	**Time (secs)**	
	Training	Test
TreeESN	0.35	0.08
GNN	101.80	0.05
RelNN	34.70	0.10

Table 7: Training and test timings (in seconds) required by TreeESN on the aggregation function tasks (averaged over the 5 tasks). Experiments on TreeESNs were run on a PC with an AMD Athlon 3000+ processor. For comparison, the timings required by GNNs and RelNNs (taken from [31]) are reported as well.

classification for the task.

The task is characterized by a large dataset containing the tree representations of 12107 documents, where each tree is obtained according to the XML structure of the corresponding document (for details see e.g. [40, 52] and references therein). The degree of the set of trees is $k = 66$ and the number of nodes in each tree varies from 3 to 115. Node labels are composed of 65 bits, only one of which is set to 1, identifying the XML tag corresponding to the node. An example of tree representation of an XML document in the INEX2006 dataset is shown in Figure 18. The INEX2006 task is a 18-class multi-classification task, accordingly the target output for each input tree is an 18-dimensional binary vector, where the unique bit equal to 1 identifies the correct classification.

structure and content datasets (see [62]). In this paper, we limit our consideration to the IEEE structure only corpus. For further details the reader is referred to [60].

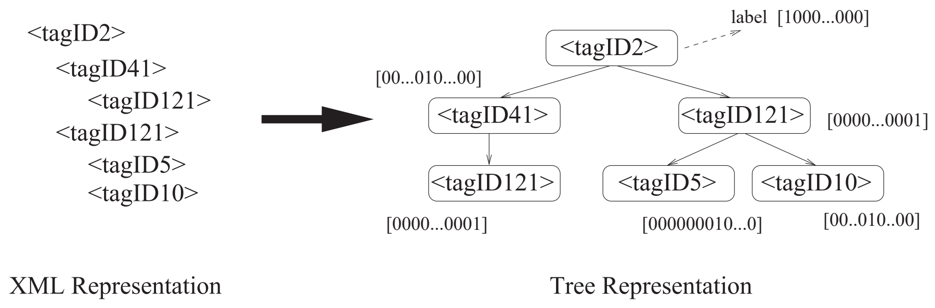


Figure 18: Example of tree representation of XML documents in the INEX2006 dataset.

The INEX2006 task has been approached by a variety of learning models, and represents a task characterized by a large number of possible output classes (with respect to other tasks considered in the paper), on which the base misclassification error achieved by a random classifier is 94.44%. This task is particularly meaningful because it allows us a performance comparison with a RecNN model extending the Self-Organizing Map (SOM) [63], called the SOM for structured data (SOM-SD) [46], whose performance in the INEX 2006 competition, although on a different version of the task, turned out to be very competitive with respect to the other approaches proposed [60, 64], and a comparison with state-of-the-art kernel models related to the same approach. Details on the application of SOM-SD to the INEX2006 classification task can be found in [52, 53]. We also take into consideration kernels for trees based on SOM-SD, namely the AM [52] and the AMπ kernel [53]. The application to the same task of other state-of-the-art kernels for tree domains, such as the PT kernel (PT) [37], the ST kernel [38], the SST kernel [39] and the Route kernel [40], is considered as well for a further significant comparison on the performance.

For experiments on TreeESN, based on the common setting described in Section 4.2, the multi-classification task was approached by training an 18-dimensional linear readout, with the output classification for a given input tree corresponding to the index of the readout unit with the largest activation. Training, validation and test sets contained 4237, 1816 and 6054 documents, respectively, as in [52, 40]. The TreeESN hyper-parametrization yielding the minimum classification error on the validation set was selected, trained on the union of the training and validation sets and tested on the test set. Table 8 shows the classification test error of TreeESNs in correspondence of the selected hyper-parametrization. In the same table the errors obtained by the SOM-SD and the kernels are presented for comparison (as obtained from the literature). Classification errors for SOM-SD, AM and AMπ correspond to different settings of the SOM-SD map (see [53]), while the errors reported for the Route kernel correspond to alternative definitions for the kernel function (see [40]).

The reservoir hyper-parametrization selected was $N_R = 500$, $\sigma = 1$, and $scale_{in} = 1$, for TreeESN-R, and $N_R = 500$, $\sigma = 3$, and $scale_{in} = 0.1$, for TreeESN-M. For both root and mean state mapping, the readout regularization selected corresponded to pseudo-inverse training (see eq. 18).

Model		**Test Error %**
TreeESN-R		57.93
TreeESN-M		57.38
SOM-SD	*mean*	64.02
	min-max	60.77-67.66
AM	*mean*	61.178
	min-max	59.93-61.77
AMπ	*mean*	60.06
	min-max	59.26-61.73
PT		58.87
ST		67.98
SST		59.56
Route	*mean*	59.02
	min-max	58.09-59.94

Table 8: Classification error on the test set for TreeESNs, SOM-SD and kernels for trees on the INEX2006 task.

Results show that TreeESNs outperform all the state-of-the-art neural and kernel-based approaches considered on this INEX2006 task for both the choices of the state mapping function. TreeESN achieved a classification error of 57.93% with standard deviation (among the guesses) of 0.20 for TreeESN-R and 57.38% with standard deviation of 0.11 for TreeESN-M. What is really noteworthy is that such performance is obtained by an extremely efficient model, whose computational cost (linear in the input size) compares well with the other approaches considered here (see Section 3.4), which include training of the recursive connections (SOM-SD), a super-linear kernel computation and the training of a SVM. For the INEX2006 task, roughly 4 minutes were required both for training and testing of TreeESN, on a PC with AMD Athlon 3200+ processor at 2.2 GHz.

TreeESNs results shown in Table 8 were obtained by using the misclassification error as evaluation measure, in order to allow a direct comparison with literature results for kernel methods on the same task. However, due to the imbalancing of the dataset [60, 61], a more expressive evaluation of the results can be obtained by considering macro and micro F1 scores, which were used as evaluation metrics for the INEX competition [60]. For a direct comparison, we refer to the work in [61], that considers a sub-set of the IEEE structure only corpus, which represent a further opportunity of comparison between TreeESNs and GNNs[6]. Such dataset, comprising the 6053 training samples in the original dataset, and referred to as the *reduced* INEX2006 dataset in the following, was organized in training, validation and test sets through a 80%-10%-10% stratified splitting (as in [61]).

We run another set of experiments on the reduced INEX2006 task considering the same range of TreeESN hyper-parametrizations used for the INEX2006 task, and adopting a balancing procedure consisting in weighting the training data with respect to the size of the target classes, analogously to [61]. Note that both the F1 scores are meaningful to evaluate the test performance, as the F1 scores were averaged over the 18 target classes with equal weights (macro-F1) and with weights dependent on class dimensions (micro-F1). TreeESN performances were obtained by model selection on macro-F1.

TreeESN-R achieved macro-F1 and micro-F1 scores of 0.3679 (with standard deviation of 0.0065) and of 0.4092 (with standard deviation of 0.0056), respectively. TreeESN-M obtained a macro-F1 of 0.3698 (with standard deviation of 0.0066) and a micro-F1 of 0.4086 (with standard deviation of 0.0061). Results obtained by GNN in [61] report a macro-F1 score of 0.48 and a micro-F1 score of 0.34. In general, it can be seen that TreeESN performance is inferior to GNN for the macro-F1 value, and superior for the micro-F1 value. Altogether, such results are interesting in particular by virtue of the extreme efficiency of the TreeESN approach. Indeed, while experiments on GNN required about 12 hours [61], training and testing a TreeESN on the reduced INEX2006 dataset required respectively $\approx$ 11 minutes and $\approx$ 1 minute (under

[6]Results achieved by GNNs [61] on the reduced dataset from the IEEE structure only corpus were the only results submitted for this task at the INEX 2006 competition.

the same conditions considered for the INEX2006 task, although networks with larger reservoirs were chosen by the model selection procedure).

4.2.3. Glycans Tasks

The analysis of the relationships between the molecular structure of glycans and their biological functions represents a challenging subject of study in the field of computational biology and an interesting area of application for Machine Learning models for structured domains [49]. In particular, several kernel methods for trees have been applied to these tasks, allowing a comparison of the TreeESN performance with state-of-the-art kernel methods for tree domains, some of which specifically tailored for the classification of glycans. In particular, we take into consideration the application to glycans classification of the Yamanishi kernel (see [50]), the ST and Subpath kernels (see [51]), baseline Kailing kernel [65], q-gram and weighted q-gram Linkage (LK), KCaM (KM), Linkage KCam (LKM) kernels (see [49]).

Glycans are carbohydrate chains supposed to take important roles in several cellular processes. Glycan molecules have complex structures that can be represented as rooted trees, where nodes stand for mono-saccharides and bonds between nodes stand for sugar bonds (see Figure 19). Roots are chosen according to the biological meaning (see [66, 67]). In this paper we consider

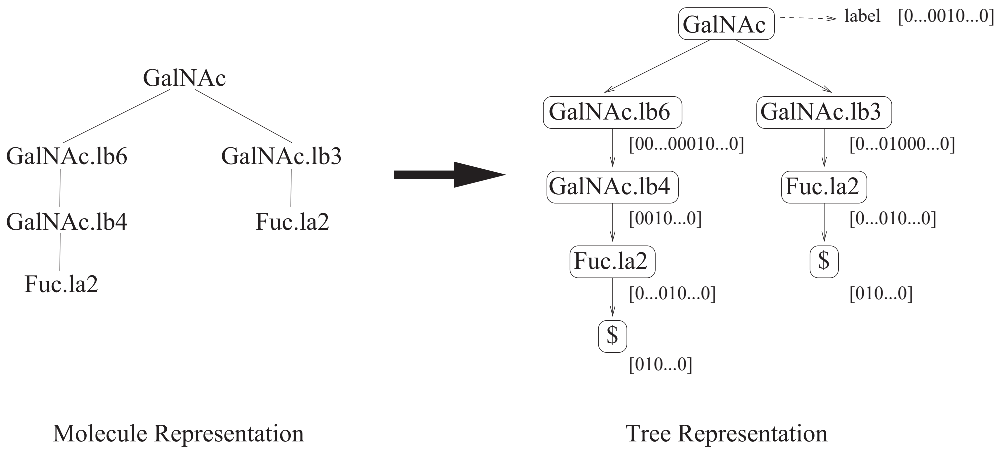


Figure 19: Example of tree representation of molecules in the Glycan datasets.

two datasets of Glycans coming from the KEGG/Glycan database [66, 67], and denoted as *Leukemia* and *Cystic*. The two datasets comprise 442 and

160 molecules, respectively. Two binary classification tasks are defined, by assigning a $+1$ target to molecules reported as leukemic or cystic fibrosis, respectively for the two datasets, and a -1 target to all the other glycan molecules. Node labels are 1-of-m encodings of the corresponding monosaccharide description, leading to a label dimension of 57 and 29 for the leukemia and cystic task, respectively. The maximum number of nodes in a tree is 23 for leukemia and 15 for cystic. For both the tasks, the maximum degree is $k = 3$.

Results were obtained by a stratified 10-fold cross validation procedure, with reservoir hyper-parametrization and readout regularization chosen on a validation set by an extra level of stratified 3-fold cross validation.

Tables 9 and 10 report the area under the curve (AUC) on the test set obtained by TreeESN-R and TreeESN-M on the Leukemia and Cystic tasks, respectively. In the same tables, the mean, maximum and minimum per-

Model		**Test AUC**
TreeESN-R		0.9493
TreeESN-M		0.9710
Yamanishi	*mean*	0.9201
	min-max	0.8970-0.9460
q-gram	*mean*	0.9340
	min-max	0.8906-0.9578
KM	*mean*	0.9352
	min-max	0.8889-0.9608
LKM	*mean*	0.9355
	min-max	0.8875-0.9623
LK	*mean*	0.9627
	min-max	0.9606-0.9647
Subpath	*mean*	0.9710
	min-max	0.9680-0.9730
ST	*mean*	0.9607
	min-max	0.9520-0.9740
Kailing	*mean*	0.9277
	min-max	0.9020-0.9530

Table 9: AUC on the test set for TreeESNs and Yamanishi, q-gram, KM, LKM, LK, Subpath, ST and Kailing kernels for trees on the Leukemia task.

Model		**Test AUC**
TreeESN-R		0.7719
TreeESN-M		0.7513
q-gram	*mean*	0.6991
	min-max	0.4794-0.8220
KM	*mean*	0.7082
	min-max	0.4980-0.8225
LKM	*mean*	0.6996
	min-max	0.4922-0.8034
LK	*mean*	0.7754
	min-max	0.7684-0.7823
Subpath	*mean*	0.8500
	min-max	0.8430-0.8560
ST	*mean*	0.7983
	min-max	0.7780-0.8090
Kailing	*mean*	0.7137
	min-max	0.6390-0.7940

Table 10: AUC on the test set for TreeESNs and q-gram, KM, LKM, LK, Subpath, ST and Kailing kernels for trees on the Cystic task.

formances of the considered kernels for glycan classification are reported for comparison, in correspondence of alternative definitions and parametrizations of the kernel functions used (for details see [50, 51, 49]).

It can be seen that the performances of TreeESNs on the two glycan classification tasks are very good and comparable to state-of-the-art results for both the choices of the state mapping function. For the Leukemia task, TreeESN-R and TreeESN-M achieved a test AUC of 0.9493 (with standard deviation of 0.0069) and of 0.9710 (with standard deviation of 0.0083), respectively. The result obtained by TreeESN-M is particularly noteworthy on this task, reaching the same performance as the Subpath kernel and outperforming those of all the other kernels considered. For the Cystic task, the test AUC obtained by TreeESN-R and TreeESN-M were respectively 0.7719 (with standard deviation of 0.0305), and 0.7513 (with standard deviation of 0.0403), which is in the range of performances of the state-of-the-art results (although inferior to the results of the Subpath and ST kernels).

Note that TreeESN results particularly efficient in comparison to the kernel methods considered in Tables 9 and 10. Indeed, as described in Section 3.4, such kernels generally involve a super-linear time complexity. An exception is the Kailing kernel which consist in the computation of simple features over the tree structures, but that shows the worsts results. On the other hand and the LK kernel that can even require an exponential time for glycan classification tasks [49]. Training and testing a TreeESN on the leukemia task approximatively required for each fold 36.84 and 3.88 seconds on average, respectively, on a PC with AMD Athlon 3000+ processor at 2 GHz. Computational times for the cystic task were of 5.98 and 0.58 seconds for training and test, respectively, on a PC with AMD Athlon 3200+ processor at 2.2 GHz.

5. Conclusions

We have presented a generalization of the RC approach to tree structured data processing, named the TreeESN model. The presented analysis would characterize the proposed RC approach in the area of neurocomputing for tree structured domains learning. TreeESNs effectively exploit the Markovian nature of contractive RecNN state dynamics, being able to discriminate among different input trees in a suffix-based fashion without any adaptation of the recurrent connections. As such, TreeESNs represent both an architectural and experimental baseline for RecNN models with trained state dynamics, and a very efficient model able to compete with more complex approaches, particularly when Markovian conditions are met in the task at hand.

For tree-to-element transductions, a fixed state mapping function is used to map the tree structured state computed by the reservoir into a vectorial feature representation that feeds the readout. In this regard, we proposed two possible choices, namely a root state mapping and a mean state mapping, which have strict relations with the Markovian properties. The effects of such relationship have been investigated through experiments on artificial ad-hoc designed tasks and on a real-world task from chemical domain, with different grade of Markovianity of the target function. In particular, the TreeESN-R, which preserves the Markovian organization of the reservoir dynamics, was effectively found to achieve a performance proportional to the degree of Markovianity of the task. More interestingly, TreeESN-M achieved promising results on complementary tasks with non-Markovian characterization: in particular, the anti-Markovian and the Alkanes tasks. TreeESN-M revealed to be a useful tool for such cases and for real-world tasks without a clear Markovian characterization. In this sense, mean state mapping can be a proper choice as an alternative to root state mapping function for recursive approaches, which was the only state mapping function used up to now for RecNN.

The evaluation of the performance, with particular emphasis on the efficiency, was investigated first in comparison to recursive neural network models, for which TreeESN is an architectural baseline, and then with respect to the wider class of kernel methods for tree data. The results confirmed a dramatic reduction of the training and test time, whenever it was possible to compare with literature results. More in general the cost of the model well compares with model including a training phase for the encoding process and with the super-linear kernel approaches. Concerning the predictive performance, the global result obtained over a variegate set of tasks in the fields of relational learning, document processing and computational biology (and using different evaluation measures) is that such efficient approach can achieve results that are comparable to those of more complex state-of-the-art neural and kernel approaches.

The issue addressed in this paper would stimulate further research in the study of effective and efficient models for learning in structured domain, and of their critical theoretical characterization. In particular, the study of TreeESN-R and TreeESN-M would open further research directions aimed at the proper extraction of state information (state mapping functions) from RC models. However, especially due to their simplicity and efficiency, the proposed models already reveal to be useful tools to approach real-world problems with complex data.

Acknowledgements

The authors would like to thank Alessandro Sperduti and Markus Hagenbuchner for providing the INEX2006 dataset.

References

[1] L. Getoor, B. Taskar, Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning), The MIT Press, 2007.

[2] L. D. Raedt, Statistical relational learning: An inductive logic programming perspective, in: Springer (Ed.), PKDD, Lecture Notes in Computer Science 3721, pp. 3–5.

[3] S. Džeroski, N. Lavrač, Relational Data Mining, Springer-Verlag, Berlin, 2001.

[4] P. Bille, A survey on tree edit distance and related problems, Theoretical Computer Science 337 (2005) 217–239.

[5] G. Xing, J. Guo, Z. Xia, Classifying xml documents based on structure/content similarity, in: N. Fuhr, M. Lalmas, A. Trotman (Eds.), Comparative Evaluation of XML Information Retrieval Systems INEX 2006, volume 4518 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2007, pp. 444–457.

[6] T. Akutsu, D. Fukagawa, A. Takasu, T. Tamura, Exact algorithms for computing the tree edit distance between unordered trees, Theoretical Computer Science 412 (2011) 352–364.

[7] P. Frasconi, M. Gori, A. Sperduti, A general framework for adaptive processing of data structures, IEEE Transactions on Neural Networks 9 (1998) 768–786.

[8] M. Diligenti, P. Frasconi, M. Gori, Hidden tree markov models for document image classification, IEEE Transactions on Pattern Analysis and Machine Intelligence 25 (2003) 519–523.

[9] N. Gianniotis, P. Tino, Visualization of Tree-Structured Data Through Generative Topographic Mapping, IEEE Transactions on Neural Networks 19 (2008) 1468–1493.

[10] D. Bacciu, A. Micheli, A. Sperduti, Bottom-up generative modeling of tree-structured data, in: Proceedings of the International Conference on Neural Information Processing (ICONIP) 2010, Springer, 2010, pp. 660–668.

[11] D. Bacciu, A. Micheli, A. Sperduti, Compositional generative mapping of structured data, in: Proceedings of the International Joint Conference on Neural Networks (IJCNN) 2010, IEEE, pp. 1–8.

[12] B. Hammer, B. Jain, Neural methods for non-standard data, in: Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2004, d-side, 2004, pp. 281–292.

[13] D. Verstraeten, B. Schrauwen, M. D'Haene, D. Stroobandt, An experimental unification of reservoir computing methods, Neural Netw. 20 (2007) 391–403.

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

[15] H. Jaeger, The "echo state" approach to analysing and training recurrent neural networks, 2001. Tech.Rep. 148, GMD - German National Research Institute for Computer Science, 2001.

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

[17] B. Hammer, P. Tiňo, Recurrent neural networks with small weights implement definite memory machines, Neural Computation 15 (2003) 1897–1929.

[18] P. Tiňo, M. Cernanský, L. Benusková, Markovian architectural bias of recurrent neural networks, IEEE Transactions on Neural Networks 15 (2004) 6–15.

[19] P. Tiňo, B. Hammer, M. Bodén, Markovian bias of neural-based architectures with feedback connections, in: Perspectives of Neural-Symbolic Integration, Springer-Verlag, 2007, pp. 95–133.

[20] C. Gallicchio, A. Micheli, Architectural and markovian factors of echo state networks, Neural Networks 24 (2011) 440 – 456.

[21] A. Sperduti, A. Starita, Supervised neural networks for the classification of structures, IEEE Transactions on Neural Networks 8 (1997) 714–735.

[22] B. Hammer, Learning with recurrent neural networks, in: Lecture Notes in Control and Information Sciences, volume 254, Springer-Verlag, 2000.

[23] B. Hammer, A. Micheli, A. Sperduti, Adaptive contextual processing of structured data by recursive neural networks: A survey of computational properties, in: Perspectives of Neural-Symbolic Integration, volume 77/2007, Springer Berlin / Heidelberg, 2007, pp. 67–94.

[24] E. Francesconi, P. Frasconi, M. Gori, S. Marinai, J. Q. Sheng, G. Soda, A. Sperduti, Logo recognition by recursive neural networks, in: Second international workshop on graphics recognition, GREC'97, Springer, 1997, pp. 104–117.

[25] A. Bianucci, A. Micheli, A. Sperduti, A. Starita, Application of cascade correlation networks for structures to chemistry, Applied Intelligence 12 (2000) 117–146.

[26] F. Costa, P. Frasconi, V. Lombardo, G. Soda, Towards incremental parsing of natural language using recursive neural networks, Applied Intelligence 19 (2003) 9–25.

[27] C. De Mauro, M. Diligenti, M. Gori, M. Maggini, Similarity learning for graph-based image representations, Pattern Recognition Letters 24 (2003) 1115 – 1122.

[28] P. Sturt, F. Costa, V. Lombardo, P. Frasconi, Learning first-pass structural attachment preferences with dynamic grammars and recursive neural networks, Cognition 88 (2003) 133–169.

[29] C. Duce, A. Micheli, A. Starita, M. Tiné, R. Solaro, Prediction of polymer properties from their structure by recursive neural networks, Macromolecular Rapid Communications 27 (2006) 711–715.

[30] W. Uwents, H. Blockeel, Classifying relational data with neural networks, in: Lecture notes in computer science, Proceedings of the 15th international conference on inductive logic programming, ILP 2005, Springer, 2005, pp. 384–396.

[31] W. Uwents, G. Monfardini, H. Blockeel, M. Gori, F. Scarselli, Neural networks for relational learning: an experimental comparison, Machine Learning 82 (2011) 315–349.

[32] F. Scarselli, M. Gori, A. Tsoi, M. Hagenbuchner, G. Monfardini, The graph neural network model, IEEE Transactions on Neural Networks 20 (2009) 61–80.

[33] B. Hammer, J. Steil, Tutorial: Perspective on learning with rnns, in: Proceedings of the European Symposium on Artificial Neural Networks,

Computational Intelligence and Machine Learning, ESANN 2002, d-side, 2002, pp. 357–368.

[34] B. Hammer, P. Tiňo, A. Micheli, A Mathematical Characterization of the Architectural Bias of Recursive Models, Technical Report 252, Universitat Osnabruck, Germany, 2004.

[35] T. Gärtner, A survey of kernels for structured data, ACM SIGKDD Explorations Newsletter 5 (2003) 49–58.

[36] D. Haussler, Convolution kernels on discrete structures, Technical Report UCSC-CRL-99-10, University of California, Santa Cruz, 1999.

[37] A. Moschitti, Efficient convolution kernels for dependency and constituent syntactic trees, in: J. Fürnkranz, T. Scheffer, M. Spiliopoulou (Eds.), Machine Learning: ECML 2006, volume 4212 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2006, pp. 318–329.

[38] S. Viswanathan, A. J. Smola, Fast kernels for string and tree matching, in: Advances in Neural Information Processing Systems 15, MIT Press, Cambridge, MA, 2003, pp. 569–576.

[39] M. Collins, N. Duffy, New ranking algorithms for parsing and tagging: kernels over discrete structures, and the voted perceptron, in: Proceedings of the Annual Meeting on Association for Computational Linguistics, ACL 2002, Association for Computational Linguistics, 2002, pp. 263–270.

[40] F. Aiolli, G. D. S. Martino, A. Sperduti, Route kernels for trees, in: Proceedings of the Annual International Conference on Machine Learning, ICML 2009, ACM, 2009, pp. 17–24.

[41] C. Gallicchio, A. Micheli, TreeESN: a preliminary experimental analysis, in: Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2010, d-side, 2010, pp. 333–338.

[42] B. Hammer, A. Micheli, A. Sperduti, M. Strickert, A general framework for unsupervised processing of structured data, Neurocomputing 57 (2004) 3 – 35.

[43] B. Hammer, A. Micheli, A. Sperduti, M. Strickert, Recursive self-organizing network models, Neural Networks 17 (2004) 1061–1085.

[44] J. Hopcroft, R. Motwani, J. Ullman, Introduction to Automata Theory, Languages, and Computation (3rd Edition), Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[45] A. Micheli, D. Sona, A. Sperduti, Contextual processing of structured data by recursive cascade correlation, IEEE Transactions on Neural Networks 15 (2004) 1396–1410.

[46] A. M. Hagenbuchner, Sperduti, A. Tsoi, A self-organizing map for adaptive processing of structured data, IEEE Transactions on Neural Networks 14 (2003) 491 – 505.

[47] Y. Bengio, P. Frasconi, P. Simard, The problem of learning long-term dependencies in recurrent networks, IEEE International Conference on Neural Networks 3 (1993) 1183–1188.

[48] J. Kolen, S. Kremer (Eds.), A Field Guide to Dynamical Recurrent Networks, IEEE Press, 2001.

[49] L. Li, W. Ching, T. Yamaguchi, K. F. Aoki-Kinoshita, A weighted q-gram method for glycan structure classification, BMC Bioinformatics 11 (2010) 33–38.

[50] Y. Yamanishi, F. Bach, J. Vert, Glycan classification with tree kernels, Bioinformatics 23 (207) 1211–1216.

[51] D. Kimura, T. Kuboyama, T. Shibuya, H. Kashima, A subpath kernel for rooted unordered trees, in: Advances in Knowledge Discovery and Data Mining, volume 6634 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2011, pp. 62–74.

[52] F. Aiolli, G. D. S. Martino, M. Hagenbuchner, A. Sperduti, Learning nonsparse kernels by self-organizing maps for structured data, IEEE Transactions on Neural Networks 20 (2009) 1938–1949.

[53] F. Aiolli, G. D. S. Martino, A. Sperduti, A new tree kernel based on som-sd, in: Proceedings of the International Conference on Articial Neural Networks, ICANN 2010, Part II, Springer, 2010, pp. 49–58.

[54] P. Tiňo, B. Hammer, Architectural bias in recurrent neural networks: Fractal analysis, Neural Computation 15 (2003) 1931–1957.

[55] A. Micheli, Neural network for graphs: A contextual constructive approach, IEEE Transactions on Neural Networks 20 (2009) 498–511.

[56] A. Micheli, F. Portera, A. Sperduti, A preliminary empirical comparison of recursive neural networks and tree kernel methods on regression tasks for tree structured domains, Neurocomputing 64 (2005) 73–92.

[57] G. K. Venayagamoorthy, B. Shishir, Effects of spectral radius and settling time in the performance of echo state networks, Neural Networks 22 (2009) 861 – 863.

[58] H. Jaeger, Reservoir riddles: suggestions for echo state network research, in: Proceedings of the IEEE International Joint Conference on Neural Networks, IJCNN 2005, volume 3, IEEE, 2005, pp. 1460–1462.

[59] D. Verstraeten, J. Dambre, X. Dutoit, B. Schrauwen, Memory versus non-linearity in reservoirs, in: Proceedings of the IEEE International Joint Conference on Neural Networks, IJCNN 2010, IEEE, 2010, pp. 2669 – 2676.

[60] L. Denoyer, P. Gallinari, Report on the xml mining track at inex 2005 and inex 2006: categorization and clustering of xml documents, SIGIR Forum 41 (2007) 79–90.

[61] S. Yong, M. Hagenbuchner, A. Tsoi, F. Scarselli, M. Gori, Xml document mining using graph neural network, in: N. Fuhr, M. Lalmas, A. Trotman (Eds.), Comparative Evaluation of XML Information Retrieval Systems, INEX 2006, volume 4518 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2007, pp. 458–472.

[62] J. D. Knijf, Fat-cat: Frequent attributes tree based classification, in: N. Fuhr, M. Lalmas, A. Trotman (Eds.), Comparative Evaluation of XML Information Retrieval Systems INEX 2006, volume 4518 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2007, pp. 485–496.

[63] T. Kohonen, Self-Organizing Maps, Springer-Verlag, 2001.

[64] M. Kc, M. Hagenbuchner, A. Tsoi, F. Scarselli, A. Sperduti, M. Gori, Xml document mining using contextual self-organizing maps for structures, in: N. Fuhr, M. Lalmas, A. Trotman (Eds.), Comparative Evaluation of XML Information Retrieval Systems, volume 4518 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2007, pp. 510–524.

[65] K. Kailing, H. Kriegel, S. Schonauer, T. Seidl, Efficient similarity search for hierarchical data in large databases, in: Advances in Database Technology, EDBT 2004, volume 2992 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2004, pp. 643–644.

[66] K. Hashimoto, S. Goto, S. Kawano, K. Aoki-Kinoshita, N. Ueda, M. H. T. Kawasaki, M. Kanehisa, Kegg as a glycome informatics resource, Glycobiology 16 (2006) 63–70.

[67] M. Kanehisa, S. Goto, S. Kawashima, Y. Okuno, M. Hattori, The kegg resource for deciphering the genome, Nucleic Acids Res. 32 (2004) 277–280.