Graph Echo State Networks

Claudio Gallicchio, *Student Member, IEEE*, and Alessio Micheli, *Member, IEEE*

Abstract— **In this paper we introduce the Graph Echo State Network (GraphESN) model, a generalization of the Echo State Network (ESN) approach to graph domains. GraphESNs allow for an efficient approach to Recursive Neural Networks (RecNNs) modeling extended to deal with cyclic/acyclic, directed/undirected, labeled graphs.**

The recurrent reservoir of the network computes a fixed contractive encoding function over graphs and is left untrained after initialization, while a feed-forward readout implements an adaptive linear output function. Contractivity of the state transition function implies a Markovian characterization of state dynamics and stability of the state computation in presence of cycles. Due to the use of fixed (untrained) encoding, the model represents both an extremely efficient version and a baseline for the performance of recursive models with trained connections.

The performance are shown on standard benchmark tasks from Chemical domains, allowing the comparison with both Neural Network and Kernel-based approaches for graphs.

I. Introduction

Reservoir Computing (RC) [1] models, and in particular Echo State Networks (ESNs) [2], [3], are Recurrent Neural Networks (RNNs) characterized by a conceptual separation between a large recurrent non-linear layer of hidden units, the *reservoir*, and a feed-forward (typically linear) layer of output units, the *readout*. Efficiency is one of the main features of this approach. Indeed, the reservoir is randomly initialized to implement a *contractive* state transition function and then is left untrained. Only the readout is adapted, e.g. by efficient linear models.

RNNs implementing contractive state dynamics have been shown to be biased toward Markovian process modeling [4], [5], being able to inherently discriminate among different input histories in a (suffix-based) Markovian flavor even before training of the recurrent connections. Such results also apply to ESNs [6], in which case training of recurrent connections is not performed at all, and Markovianity represents a distinctive characterization of the model.

Recursive Neural Networks (RecNNs) [7], [8] generalize RNNs to structured domains, allowing for an adaptive processing of structured data in many real-world applicative domains [8]. RecNNs are based on an encoding process in which a state representation is recursively computed according to the topological relationships among the vertices in the input structure. RecNNs constitute a very powerful class of models, with interesting computational properties [9]. However, problems related to training of RNNs in general transfer to RecNNs as well, and training RecNNs may be even more computationally expensive than training RNNs. Thereby, it is worth investigating efficient approaches to RecNNs modeling. Recently, an extension of ESNs to tree structured domains, the Tree Echo State Network (TreeESN) model [10] has been proposed for efficiently processing tree structured data. A contractive setting of the state transition function in RecNN models induces a Markovian characterization [11] of state dynamics which applies to TreeESNs as well. Thereby different input structures sharing a common suffix (extended to the concept of top sub-tree starting from the root) are mapped into states which are similar to each other proportionally to the similarity of such suffix.

Claudio Gallicchio and Alessio Micheli are with the Department of Computer Science, University of Pisa, Largo Bruno Pontecorvo 3, 56127 Pisa, Italy (email: {gallicch,micheli}@di.unipi.it).

Another critical aspect to consider concerns the class of data structures supported. RecNNs in general, and TreeESNs in particular, naturally deal with hierarchical structured information and do not deal with general graphs. In particular, RecNNs are suitable for processing domains of directed acyclic graphs, while in case of cyclic dependencies in the state computations (which may occur e.g. in the case of directed cyclic graphs or undirected graphs processing) the stability of the encoding process is not guaranteed [7]. Recently, two different approaches have been proposed to overcome this limitation. The Neural Networks for Graphs (NN4G) model [12] is based on a constructive feed-forward architecture that eliminates the need of cyclic dependencies among state variables. On the other hand, the Graph Neural Network (GNN) [13] is a RecNN model trained using a gradient descent-based learning algorithm in which a phase of state relaxation and a phase of gradient computation are alternated. In this case, stability of the encoding process (and consequently convergence of the relaxation phase) is obtained by constraining the cost function in order to achieve a condition of contractivity on the state transition function. Thereby a Markovian characterization of global state dynamics follows for GNNs as well and opens the issue of investigating efficient alternatives that explicitly exploits the Markovian characterization. Indeed, using the strategy adopted by GNN, the processing of cyclic graphs is paid in terms of efficiency of the learning algorithm.

In this paper, we propose an extension of the ESN (and TreeESN) approach to process graph domains, named the Graph Echo State Network (GraphESN). GraphESNs consist in a hidden reservoir layer of non-linear recursive units, responsible for the encoding process, plus a readout layer of linear feed-forward units. The reservoir is initialized to implement a contractive state transition function and then is left untrained, while the readout may be trained e.g. by using efficient linear models. The condition of contractivity, inherited from ESNs and TreeESNs, assumes in this case a relevant specific meaning in terms of the extension of the class of data structures supported. Indeed, contractivity of GraphESN dynamics implies stability of state computation

978-1-4244-8126-2/10/$26.00 ©2010 IEEE

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on March 13,2021 at 05:31:35 UTC from IEEE Xplore. Restrictions apply.

also in case of cyclic dependencies and thus allows to extend the applicability of the efficient RC approach to a large class of cyclic/acyclic, directed/undirected, labeled graphs. Moreover, being characterized by *fixed* contractive state dynamics, the GraphESN may represent an architectural baseline for other RecNNs implementing adaptive contractive state transition functions, such as GNNs. In this concern, it is interesting to empirically evaluate the effective distance occurring between the performance of methods based on fixed or adaptive encoding under the contractivity constraint. On the other hand, experimental results of GraphESNs, based on the inherent ability to discriminate among graph structures in absence of learning of recurrent connections, represent a baseline performance for the class of RecNN models with trained recurrent connections.

II. Transductions on Graph Domains

A graph $\mathbf{g}$ in a set of graphs $\mathcal{G}$ is a couple $(V(\mathbf{g}), E(\mathbf{g}))$, where $V(\mathbf{g})$ denotes the set of vertices of $\mathbf{g}$ and $E(\mathbf{g}) = \{(u,v) : u,v \in V(\mathbf{g})\}$ denotes the set of edges of $\mathbf{g}$. The number of vertices of $\mathbf{g}$ is denoted by $|V(\mathbf{g})|$. For directed graphs, each edge has a direction (i.e. (u,v) leaves vertex u and enters vertex v). In the following, we use V and E to indicate the set of vertices and the set of edges, respectively, whenever the reference to the graph is implicit. The neighborhood of a vertex v in a undirected graph is the set of adjacent vertices of v, i.e. $\mathcal{N}(v) = \{u \in V : (u,v) \in E\}$. For directed graphs, the neighbors of v can be distinguished in predecessors of v (i.e. $\mathcal{P}(v) = \{u : (u,v) \in E\}$) and successors of v (i.e. $\mathcal{S}(v) = \{u : (v,u) \in E\}$). In this case $\mathcal{N}(v) = \mathcal{P}(v) \cup \mathcal{S}(v)$. The degree of vertex v, i.e. the dimension of the neighborhood of v, is indicated by $degree(v) = |\mathcal{N}(v)|$, while the maximum degree assumed over the set of graphs $\mathcal{G}$ is denoted by k.
Each vertex v has a vectorial numerical label associated, $\mathbf{u}(v) \in \mathbb{R}^{N_U}$, where $\mathbb{R}^{N_U}$ denotes a vectorial label input space. The set of graphs with vertex labels in $\mathbb{R}^{N_U}$ is indicated by $(\mathbb{R}^{N_U})^{\#}$.

We are interested in computing structural transductions on graph domains. A *structural transduction* $\mathcal{T}$ is a function mapping an input graph domain $(\mathbb{R}^{N_U})^{\#}$ into an output graph domain $(\mathbb{R}^{N_O})^{\#}$:

$$\mathcal{T} : (\mathbb{R}^{N_U})^{\#} \rightarrow (\mathbb{R}^{N_O})^{\#} \tag{1}$$

In the following, $\mathbf{y}(v) \in \mathbb{R}^{N_O}$ denotes the vectorial label associated to a vertex v of a graph in the output structured domain. For a *structure-to-structure* (also known as node-focused [13]) transduction $\mathcal{T}$, the output graph is isomorphic to the input graph, i.e. $\mathcal{T}$ associates an output vertex in correspondence of each input vertex (equivalently, a vectorial output value is computed in correspondence of each vertex of the input structure). In a *structure-to-element* (also known as graph-focused [13]) transduction $\mathcal{T}$, a single vectorial output value is computed for the whole input structure (in this case the structured output space reduces to a vectorial real subspace). $\mathcal{T}$ is a *stationary* transduction if the function it computes does not depend on the particular vertex to which it is applied. An *adaptive* transduction is learned from observed data and not a-priori fixed.
A structural transduction can be usefully decomposed as $\mathcal{T} : \mathcal{T}_{out} \circ \mathcal{T}_{enc}$, where $\mathcal{T}_{enc}$ is the *encoding function* and $\mathcal{T}_{out}$ is the *output function*. The encoding function $\mathcal{T}_{enc} : (\mathbb{R}^{N_U})^{\#} \rightarrow (\mathbb{R}^{N_R})^{\#}$ maps the input structured domain into a feature structured domain $(\mathbb{R}^{N_R})^{\#}$, where $\mathbb{R}^{N_R}$ denotes the vectorial feature space. The output function $\mathcal{T}_{out} : (\mathbb{R}^{N_R})^{\#} \rightarrow (\mathbb{R}^{N_O})^{\#}$ maps the structured feature representation of the input graph into the output structured representation. In order to realize structure-to-element transductions, in the following we resort to a *state mapping function* $\mathcal{X} : (\mathbb{R}^{N_R})^{\#} \rightarrow \mathbb{R}^{N_R}$. $\mathcal{X}$ is preliminary applied to the output of the encoding function, to obtain a single fixed-size feature representation for the whole input graph. In this case, $\mathcal{T} = \mathcal{T}_{out} \circ \mathcal{X} \circ \mathcal{T}_{enc}$.

Considering a supervised learning paradigm, a training set of input graphs is represented by $\mathfrak{T} = \{(\mathbf{g}, \bar{\mathbf{y}}(\mathbf{g})) : \mathbf{g} \in \mathcal{G}, \bar{\mathbf{y}}(\mathbf{g}) \in (\mathbb{R}^{N_O})^{\#}\}$, where $\bar{\mathbf{y}}(\mathbf{g})$ is the target output associated to graph $\mathbf{g}$. For structure-to-element transductions, $\bar{\mathbf{y}}(\mathbf{g})$ is a fixed-size vector in $\mathbb{R}^{N_O}$. For instance, in the case of binary classification tasks, $\bar{\mathbf{y}}(\mathbf{g}) \in \{-1, 1\}$.

Recursive approaches, including the proposed GraphESN, are dynamical systems realizing the encoding $\mathcal{T}_{enc}$, for which the feature space $(\mathbb{R}^{N_R})^{\#}$ is composed by state variables associated with each vertex of the input graph. In the following, the state information associated to vertex v of graph $\mathbf{g}$ is represented as $\mathbf{x}(v) \in \mathbb{R}^{N_R}$. The concatenation of the states associated to each vertex in the neighborhood of v is denoted by $\mathbf{x}(\mathcal{N}(v)) \in \mathbb{R}^{kN_R}$, with k equal to the maximum degree over the set of graphs. Note that $\mathbf{x}(\mathcal{N}(v))$ is arranged according to the properties of the set of graphs under consideration. In particular, for directed graphs $\mathbf{x}(\mathcal{N}(v))$ is organized such that predecessors and successors of v are distinguished (e.g. the concatenation contains the states of predecessors of v followed by the states of successors of v). If the neighborhood of v contains $m < k$ vertices, a null state $\mathbf{x}(nil) = \mathbf{0} \in \mathbb{R}^{N_R}$ is used in correspondence of the undefined neighbors. The concatenation of the states associated to every vertex in an input graph $\mathbf{g}$, according to an arbitrary ordering of the vertices of $\mathbf{g}$, is denoted by $\mathbf{x}(\mathbf{g}) \in \mathbb{R}^{|V(\mathbf{g})|N_R}$.

For the class of recursive transductions considered in the following, for every input graph $\mathbf{g}$, the encoding function $\mathcal{T}_{enc}$ associates a state information to every vertex $v \in V(\mathbf{g})$ according to a *local encoding function* τ:

$$\mathbf{x}(v) = \tau(\mathbf{u}(v), \mathbf{x}(\mathcal{N}(v))) \tag{2}$$

which expresses the dependency of the state of vertex v on the states of the neighbors of v. Equivalently, $\mathcal{T}_{enc}$ associates to an input graph $\mathbf{g}$, a global state information $\mathbf{x}(\mathbf{g})$, according to a *global encoding function* $\hat{\tau}$ consisting in the application of equation 2 to every vertex in $V(\mathbf{g})$:

$$\mathbf{x}(\mathbf{g}) = \hat{\tau}(\mathbf{g}, \mathbf{x}(\mathbf{g})) \tag{3}$$

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on March 13,2021 at 05:31:35 UTC from IEEE Xplore. Restrictions apply.

Equation 3 expresses all the dependencies of the states of the vertices of $\mathbf{g}$ on the states of their respective neighbors. Given an input graph $\mathbf{g}$, the output of the encoding process $\mathcal{T}_{enc}$ corresponds to a solution of equation 3.
Unfortunately, in the case of mutual dependencies among states of different vertices, which might occur for directed cyclic graphs and for undirected graphs [1] as input domain structures, the existence of a solution of equation 3 might not be guaranteed. However, the Banach Contraction Principle [14] ensures the existence and uniqueness of a solution of equation 3 whenever $\hat{\tau}$ is *contractive* with respect to the state $\mathbf{x}(\mathbf{g})$. Under such hypothesis, the dynamical system ruled by $\hat{\tau}$ will always converge to the fixed point of equation 3 under any initial condition on $\mathbf{x}(\mathbf{g})$. One simple way to compute a stable encoding $\mathbf{x}(\mathbf{g})$, therefore relies on the application of an iterative version of equation 2 to every vertex in the input graph, under the condition of contractivity of $\hat{\tau}$ in equation 3. This can be easily and efficiently implemented by an extension to graph domains of the ESN approach, named the GraphESN model.

III. The Graph Echo State Network Model

A GraphESN computes stationary partially adaptive transductions on graph structured domains. It consists in an input layer of N_U units, a hidden layer of N_R non-linear recursive units (the reservoir), and an output layer of N_O linear feed-forward units (the readout).

The *reservoir* is responsible for computing a fixed recurrent encoding function, while the *readout* computes an adaptive feed-forward output function. For structure-to-element transductions, a *state mapping function* is used to obtain a single fixed-size state information. The following subsections describe these components more in depth.

A. The Reservoir

The encoding function $\mathcal{T}_{enc}$ is computed by the reservoir of the GraphESN implementing an iterative version of τ in equation 2. For each iterative step t, the *local state transition function* of the reservoir computes for every vertex v a state value $\mathbf{x}_t(v)$ according to the following equation:

$$\begin{aligned}\mathbf{x}_t(v) = \tau(\mathbf{u}(v), \mathbf{x}_{t-1}(\mathcal{N}(v))) = \\ f(\mathbf{W}_{in}\mathbf{u}(v) + \hat{\mathbf{W}}_{\mathcal{N}}\mathbf{x}_{t-1}(\mathcal{N}(v)))\end{aligned} \tag{4}$$

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), $\hat{\mathbf{W}}_{\mathcal{N}} \in \mathbb{R}^{N_R \times kN_R}$ is the recurrent weight matrix for the states of the neighbors of v and f is the activation function of the reservoir units. The initial state for each $v \in V$ is defined as the null state: $\mathbf{x}_0(v) = \mathbf{0} \in \mathbb{R}^{N_R}$.
The recurrent weight matrix $\hat{\mathbf{W}}_{\mathcal{N}}$ is organized in k sub-matrices with dimension $N_R \times N_R$, arranged accordingly to the characteristics of the set of graphs considered. For

[1]In an undirected graph, each edge can be interpreted in terms of a couple of directed edges composing a cycle involving the two mutually connected vertices.

```
- For each $\mathbf{g} \in \mathcal{G}$
-   Initialize $t = 0$
-   For each $v \in V(\mathbf{g})$
-     Initialize $\mathbf{x}_0(v) = \mathbf{0} \in \mathbb{R}^{N_R}$
-   Repeat
-     $t = t + 1$
-     For each $v \in V(\mathbf{g})$ [any visiting order]
-         Compute $\mathbf{x}_t(v) =$ equation 4
-   Until $\|\mathbf{x}_t(\mathbf{g}) - \mathbf{x}_{t-1}(\mathbf{g})\|_2 \leq \epsilon$
-         [convergence of $\mathbf{x}(\mathbf{g})$, see text]
```

Fig. 1. Algorithm for the iterative computation of the global state in a GraphESN.

directed graphs, different sub-matrices can be used in correspondence of predecessors and successors, while for positional graphs, different sub-matrices can be used in correspondence of different positions in the neighborhood. The concatenation of the states computed at time step t for all the vertices in the input graph $\mathbf{g}$ is denoted by $\mathbf{x}_t(\mathbf{g})$.

Remark 1: In standard ESNs, used to process sequential input patterns, a time step index, usually denoted by t, is associated to each element (i.e. vertex) of the input sequence. The serial ordered application of the state transition function for each time step t defines a visit of the input sequence. For GraphESNs the visiting process of the input graph is ruled by the variable v denoting the vertices. Note also that the index t is used here to denote the t-th iterate of the computation of the same $x(v)$, i.e. it is the index used to rule the relaxation of the equation 2 and it is not related to the input data. ■

The processes of visiting a graph and of convergence of the global state of a GraphESN are synthesized in the algorithm in Figure 1.

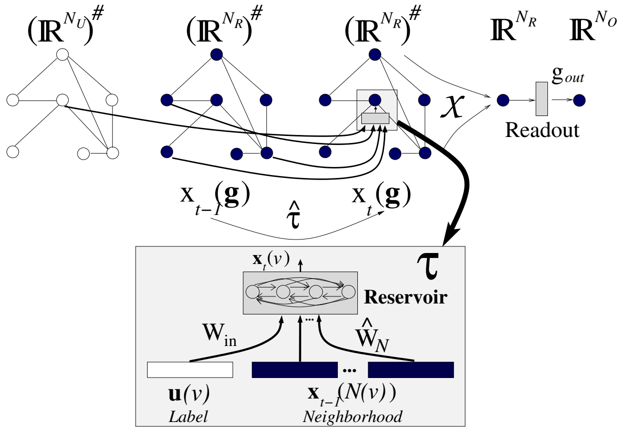

Fig. 2. An input graph, one pass of the encoding process computed by the reservoir in the state representation, the state mapping function and the output function for a GraphESN.

Figure 2 shows an input graph in $(\mathbb{R}^{N_U})^\#$, one pass of the encoding process in the structured (feature) state space $(\mathbb{R}^{N_R})^\#$, the state mapping function $\mathcal{X}$ and the vectorial output in the output space $\mathbb{R}^{N_O}$. The description of the step of the encoding process, in the structured state representation isomorphic to the input graph, shows the reservoir architecture corresponding to equation 4 applied to one single

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on March 13,2021 at 05:31:35 UTC from IEEE Xplore. Restrictions apply.

vertex v. By the stationary assumption, the same architecture is applied, for each iterative step of the encoding process, to every vertex in the graph. Note that in this process the number of vertices in the input graph $|V(\mathbf{g})|$ and the dimension of the reservoir N_R are independent of each other.

To the only aim of defining a condition for the initialization of the reservoir (according to a constraint on contractivity of the global encoding function $\hat{\tau}$ compatible with [13]), for each vertex $v \in V(\mathbf{g})$ we denote by $\hat{\mathbf{W}}_v \in \mathbb{R}^{N_R \times |V(\mathbf{g})| N_R}$ the *global* recurrent weight matrix obtained by arranging $\hat{\mathbf{W}}_{\mathcal{N}}$ and a number of $|V(\mathbf{g})| - k$ zero matrices, $\mathbf{0} \in \mathbb{R}^{N_R \times N_R}$, such that for every v in $V(\mathbf{g})$, equation 4 can be re-formulated as follows:

$$\mathbf{x}_t(v) = f(\mathbf{W}_{in}\mathbf{u}(v) + \hat{\mathbf{W}}_v \mathbf{x}_{t-1}(\mathbf{g})) \tag{5}$$

Note that the matrices $\hat{\mathbf{W}}_v$, for every v in $V(\mathbf{g})$, are all the same up to a number of column permutations. The application of the local state transition function of equation 5 to every vertex v in $V(\mathbf{g})$ defines the computation of an iterative version of $\hat{\tau}$ in equation 3, called the *global state transition function*. The global state transition function computes the new global state for the whole input structure $\mathbf{g}$ given the input labels for all the vertices of $\mathbf{g}$ and the global state at the previous time step of computation:

$$\begin{array}{c} \mathbf{x}_t(\mathbf{g}) = \hat{\tau}(\mathbf{g}, \mathbf{x}_{t-1}(\mathbf{g})) = \\ \left(\begin{array}{c} f(\mathbf{W}_{in}\mathbf{u}(v_1) + \hat{\mathbf{W}}_{v_1}\mathbf{x}_{t-1}(\mathbf{g})) \\ \cdots \\ f(\mathbf{W}_{in}\mathbf{u}(v_{|V(\mathbf{g})|}) + \hat{\mathbf{W}}_{v_{|V(\mathbf{g})|}}\mathbf{x}_{t-1}(\mathbf{g})) \end{array} \right) \end{array} \tag{6}$$

As in standard ESNs, the weight values of the recurrent connections in a GraphESN are initialized according to a contractive condition on the global state transition function and then are left untrained. Thereby, $\exists C \in (0,1)$ such that $\forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^{|V(\mathbf{g})| N_R}$, $\forall \mathbf{g} \in (\mathbb{R}^{N_U})^{\#}$: $\|\hat{\tau}(\mathbf{g}, \mathbf{x}) - \hat{\tau}(\mathbf{g}, \mathbf{x}')\| \leq C\|\mathbf{x} - \mathbf{x}'\|$ must hold. A simple sufficient condition for the contractivity in the Euclidean norm of the global state transition function in the case of *tanh* activation function is derived as follows:

$$\begin{array}{l} \forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^{|V(\mathbf{g})| N_R}, \forall \mathbf{g} \in (\mathbb{R}^{N_U})^{\#} : \|\hat{\tau}(\mathbf{g}, \mathbf{x}) - \hat{\tau}(\mathbf{g}, \mathbf{x}')\|_2 \\ = \| \left(\begin{array}{c} tanh(\mathbf{W}_{in}\mathbf{u}(v_1) + \hat{\mathbf{W}}_{v_1}\mathbf{x}) \\ \cdots \\ tanh(\mathbf{W}_{in}\mathbf{u}(v_{|V(\mathbf{g})|}) + \hat{\mathbf{W}}_{v_{|V(\mathbf{g})|}}\mathbf{x}) \end{array} \right) \\ - \left(\begin{array}{c} tanh(\mathbf{W}_{in}\mathbf{u}(v_1) + \hat{\mathbf{W}}_{v_1}\mathbf{x}') \\ \cdots \\ tanh(\mathbf{W}_{in}\mathbf{u}(v_{|V(\mathbf{g})|}) + \hat{\mathbf{W}}_{v_{|V(\mathbf{g})|}}\mathbf{x}') \end{array} \right) \|_2 \\ \leq \sum_{v \in V(\mathbf{g})} (\|tanh(\mathbf{W}_{in}\mathbf{u}(v) + \hat{\mathbf{W}}_v\mathbf{x}) - \\ \qquad tanh(\mathbf{W}_{in}\mathbf{u}(v) + \hat{\mathbf{W}}_v\mathbf{x}')\|_2) \\ \leq \sum_{v \in V(\mathbf{g})} (\|\hat{\mathbf{W}}_v(\mathbf{x} - \mathbf{x}')\|_2) \leq \sum_{v \in V(\mathbf{g})} (\|\hat{\mathbf{W}}_v\|_2 \|\mathbf{x} - \mathbf{x}'\|_2) \end{array}$$

Note that $\|\hat{\mathbf{W}}_v\|_2 = \|\hat{\mathbf{W}}_{\mathcal{N}}\|_2$ for all $v \in V(\mathbf{g})$, as $\hat{\mathbf{W}}_v$ is obtained through column permutations on $[\mathbf{W}_{\mathcal{N}}\mathbf{0}\ldots\mathbf{0}]$. Therefore, it follows that:

$$\begin{array}{l} \forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^{|V(\mathbf{g})| N_R}, \forall \mathbf{g} \in (\mathbb{R}^{N_U})^{\#} : \|\hat{\tau}(\mathbf{g}, \mathbf{x}) - \hat{\tau}(\mathbf{g}, \mathbf{x}')\|_2 \\ \leq \sum_{v \in V(\mathbf{g})} (\|\hat{\mathbf{W}}_{\mathcal{N}}\|_2 \|\mathbf{x} - \mathbf{x}'\|_2) = |V(\mathbf{g})| \|\hat{\mathbf{W}}_{\mathcal{N}}\|_2 \|\mathbf{x} - \mathbf{x}'\|_2 \end{array}$$

Thus, contractivity of the global state transition function $\hat{\tau}$ is always ensured whenever

$$\sigma = (\max_{\mathbf{g}} |V(\mathbf{g})|) \ \ \|\hat{\mathbf{W}}_{\mathcal{N}}\|_2 \ < \ 1 \tag{7}$$

holds, where σ is called the *contraction coefficient* of the network and $\max_{\mathbf{g}} |V(\mathbf{g})|$ represents the maximum number of vertices in a graph considered. Similarly to the Echo State Property (ESP), equation 7 implies a very simple procedure to obtain a valid reservoir: matrices $\mathbf{W}_{in}$ and $\hat{\mathbf{W}}_{\mathcal{N}}$ are randomly initialized, then $\hat{\mathbf{W}}_{\mathcal{N}}$ is simply re-scaled such that $\sigma < 1$ holds.

The computation of the global state transition function $\hat{\tau}$ for each graph $\mathbf{g}$ is iteratively repeated until convergence to the unique fixed point of the dynamical system ruled by $\hat{\tau}$. By initialization of $\hat{\mathbf{W}}_{\mathcal{N}}$ according to equation 7, the convergence of the reservoir computation for every initial global state $\mathbf{x}_0(\mathbf{g})$ is always ensured by the Banach Contraction Principle. In practice, the global state transition function is iterated until the distance between two consecutive global states $\mathbf{x}_t(\mathbf{g})$ and $\mathbf{x}_{t+1}(\mathbf{g})$ is smaller than a threshold ϵ (see algorithm in Figure 1).

Contractivity of the global state transition function $\hat{\tau}$ characterizes GraphESN dynamics under several points of view. First, cyclic dependencies in the local state computations, which represent a problem for standard RecNN models [7], can naturally be managed by GraphESNs because of the stability of the global state representation. Second, stability of the state representation of the encoding process for every input graph also implies a property of GraphESNs similar to the ESP of standard ESNs. For every input graph $\mathbf{g}$, the global state computed by the reservoir, i.e. $\mathbf{x}(\mathbf{g})$, (definitely) depends only on $\mathbf{g}$ itself, while any dependency on the initial global state is progressively lost. In this case, the transient period simply consists in the iterative application of the global state transition function until convergence.

Third, contractivity of the global state transition function implies a Markovian characterization of the state dynamics, as in standard ESNs [6] and in TreeESNs [10]. The impact of Markovianity have been so far investigated only for recurrent models on sequence domains [4], for standard ESNs [15], [6] and for tree domains [11], [10]. Intuitively, the organization of states for GraphESN generalizes the suffix-based state organization found for Markovian models on sequences and trees. The concept of suffix on sequences and trees can be extended with the concept of common set of d-neighbors local to each vertex v, i.e. the sub-graph induced by vertices at distance $h \leq d$ from v (see Figure 3). Figure 3 reports also the state information flow of over $(\mathbb{R}^{N_R})^{\#}$ for the computation of $\mathbf{x}(v)$, showing the states in $(\mathbb{R}^{N_R})^{\#}$ which are indirectly involved through the iterative application of equation 4, up to an incremental distance d.

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on March 13,2021 at 05:31:35 UTC from IEEE Xplore. Restrictions apply.

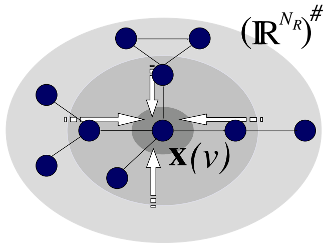

Fig. 3. Vertices up to distance $d = 2$ from v (i.e. up to d-neighbors) and the information flow over $(\mathbb{R}^{N_R})^{\#}$ to compute the state $\mathbf{x}(v)$.

The Markovian analysis, which deserves further studies, can be relevant to characterize the behavior and the limitations of models for graph domains based on recursive contractive dynamics, both for fixed and adaptive state encodings, i.e. for both the cases of models in which learning of the state transition function is implemented (like GNN) or not (like GraphESN). In particular, we hypothesize that the assumption of contractivity can have a major role beyond the architectural details. As a first step, in the following we empirically evaluate the effects of fixed contractive state dynamics on real-world tasks with graph patterns. This allows us to assess the relevance of the assumption of contractivity on such tasks and to investigate the use of GraphESNs as reliable baseline of the class of recursive models.

B. The State Mapping Function

In case of structure-to-element transductions applied to an input graph $\mathbf{g}$, after the convergence of the reservoir computation to a fixed global state $\mathbf{x}(\mathbf{g})$, the state mapping function $\mathcal{X}$ computes a fixed-size N_R-dimensional state representation for $\mathbf{g}$, i.e. $\mathcal{X}(\mathbf{x}(\mathbf{g})) \in \mathbb{R}^{N_R}$. Here we consider two choices for the computation of the state mapping function.
A *supersource state mapping* (also known as root state mapping in [10]) consists in mapping $\mathbf{x}(\mathbf{g})$ into the state of the supersource of $\mathbf{g}$. The root state mapping can be applied to structures for which a supersource is defined. Otherwise a preprocessing of the input structures is required as in [7].
A *mean state mapping* computes the fixed-size state $\mathcal{X}(\mathbf{x}(\mathbf{g}))$ as the mean of the states computed for the vertices of $\mathbf{g}$:

$$\mathcal{X}(\mathbf{x}(\mathbf{g})) = \frac{1}{|V(\mathbf{g})|} \sum_{v \in V(\mathbf{g})} \mathbf{x}(v) \tag{8}$$

C. The Readout

The readout part of the network computes the adaptive output function by resorting to a *local output function* g_{out}. In the simplest case, function g_{out} is implemented through a layer of N_O linear units.

For structure-to-structure transductions, after the convergence of the reservoir computation, the local output function is applied to the state of each vertex v of $\mathbf{g}$:

$$\mathbf{y}(v) = g_{out}(\mathbf{x}(v)) = \mathbf{W}_{out}\mathbf{x}(v) \tag{9}$$

where $\mathbf{y}(v) \in \mathbb{R}^{N_O}$ is the output (vector) value computed for vertex v and $\mathbf{W}_{out} \in \mathbb{R}^{N_O \times N_R}$ is the reservoir-to-readout weight matrix (possibly including a bias term). The application of g_{out} to the state of every vertex of the input graph defines the computation of the output function $\mathcal{T}_{out}$.

For structure-to-element transduction, the fixed-size output for the whole input graph $\mathbf{g}$, denoted by $\mathbf{y}(\mathbf{g})$, is obtained by applying the local output function to the state returned by the state mapping function $\mathcal{X}$:

$$\mathbf{y}(\mathbf{g}) = g_{out}(\mathcal{X}(\mathbf{x}(\mathbf{g}))) = \mathbf{W}_{out}\mathcal{X}(\mathbf{x}(\mathbf{g})) \tag{10}$$

As in standard ESNs, training of the readout is performed by adjusting weight values in $\mathbf{W}_{out}$ through linear regression. Therefore, given a training set $\mathfrak{T}$, containing a number of P structured input patterns, the states computed by the reservoir after the encoding process are column-wise arranged into a state matrix $\mathbf{X} \in \mathbb{R}^{N_R \times P}$. Analogously, the target outputs for the training patterns are column-wise arranged into a matrix $\bar{\mathbf{Y}} \in \mathbb{R}^{N_O \times P}$. Matrix $\mathbf{W}_{out}$ is then selected to minimize

$$\|\mathbf{W}_{out}\mathbf{X} - \bar{\mathbf{Y}}\|_2^2 \tag{11}$$

D. Computational Complexity of the Model

For each input graph $\mathbf{g}$, one step of the encoding process consists in the application of equation 4 for each vertex $v \in V(\mathbf{g})$, requiring a number of $O(|V(\mathbf{g})|\ k\ N_R^2)$ operations. This cost is bounded by $O(|V(\mathbf{g})|^2\ N_R^2)$ corresponding to the case of fully connected (complete) graphs. However, in typical cases of learning with structured data, e.g. in Chemical domains, graphs are sparse and often with a fixed value of k (e.g. $k = 4$ for the datasets considered in our experiments). Moreover, if the reservoir is sparsely connected as in standard ESNs, such that each reservoir unit is connected to at most M other units, the cost of one pass of the encoding process reduces to

$$O(|V(\mathbf{g})|\ k\ N_R\ M) \tag{12}$$

which linearly scales with both the number of vertices in the input graph and the number of units in the reservoir. The Banach Contraction Principle ensures that the convergence of the encoding process is fast and in practice a maximum (small) number of iterations could be fixed (e.g. in our experiments convergence of the encoding process required at most 13 iterations).
Note that the cost of the encoding process in GraphESNs is the same for both the training and the test phase, resulting in a very efficient strategy. For the sake of comparison, the encoding process in GNNs [13] during training may require several hundreds or thousands epochs, and for each epoch the computational cost is given by the sum of the cost of the convergent iterative computation of the stable state (i.e. the whole encoding process in a GraphESN) and the cost of the gradient computation.
The computational cost of the encoding process in GraphESNs compares well also with kernel-based methods for structured domains. For instance, under the same assumption of a bounded maximum degree k, the cost of the optimal assignment kernel and the expected match kernel, proposed in [16] and considered in Section IV for comparison of

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on March 13,2021 at 05:31:35 UTC from IEEE Xplore. Restrictions apply.

experimental results, is respectively cubic and quadratic in the number of vertices of the input graph.

The cost of training the linear readout in a GraphESN depends on the method used to solve the linear regression problem of equation 11. This can range from a direct method using singular value decomposition of matrix $\mathbf{X}$, whose cost is cubic in the number of patterns, to efficient iterative approaches in which case the cost of each epoch is linear with the number of patterns.
Note that the output function in GraphESNs is implemented through an extremely simple readout, i.e. just a layer of feed-forward linear units, and the cost of its training procedure is generally inferior to the cost of training more complex readout implementations, such as MLPs (as in GNN [13]) or SVMs (as in kernel-based methods, e.g. in [16], [17], [18]).

IV. Experiments

We applied the GraphESN model on two tasks from a Chemical domain related to the analysis of toxicity of chemical compounds. We considered the well known Mutagenesis dataset [19] and the Predictive Toxicology Challenge (PTC) dataset [20], on which the results of GraphESN may represent a baseline performance for other approaches with learning. In particular, the Mutagenesis dataset allows us to empirically evaluate the effective distance occurring between the performance of methods based on a fixed or adaptive contractive encoding. Moreover, this dataset allows a comparison with some ILP methods for structured data. On the other hand, the PTC dataset allows comparisons with a large class of learning models including kernel-based methods for structured domains.

Mutag. Dataset. The well known Mutagenesis dataset [19] contains a description of 230 nitroaromatic molecules and the goal consists in recognizing the mutagenic compounds. We considered the *whole* Mutagenesis dataset. Each molecule is described by its atom-bond structure (AB), two chemical measurements (C) and two precoded structural attributes (PS). We considered the three possible descriptions AB, AB+C and AB+C+PS. For each atom in a molecule, the AB description specifies element, atom type and partial charge. The maximum number of atoms in a molecule is 40 and the maximum degree is 4.
In our experiments, each molecule is represented as an undirected graph in which vertices correspond to atoms and edges correspond to bonds. Each vertex label contains a 1-of-m bipolar encoding of the element of the corresponding atom, its partial charge, its atom type normalized in $[-1, 1]$, and global C and PS attributes (when considered). The total dimension of the label was 11, 13 and 15 for the AB, AB+C, AB+C+PS descriptions, respectively. The task consists in a binary classification, where the target of each graph is 1 if the corresponding molecule is mutagenic and -1 otherwise.

PTC Dataset. The PTC dataset [20] reports the carcinogenicity of 417 chemical compounds relatively to four types of rodents: male rats (MR), female rats (FR), male mice (MM) and female mice (FM). The maximum number of atoms in a molecule is 109 and the maximum degree is 4. Each molecule is represented as an undirected graph, with vertices corresponding to atoms and edges corresponding to bonds. The label attached to each vertex contains a 1-of-m bipolar encoding of the element of the corresponding atom and its partial charge, yielding a total dimension of the label of 23. A binary classification task is defined for each type of rodents, i.e. MR, FR, MM and FM. For each task, the target class of each graph is 1 if the corresponding molecule is active and -1 otherwise according to the schema in [16].

A. Experimental Settings

GraphESNs considered in experiments were initialized as follows. Weight values in matrix $\mathbf{W}_{in}$ were initialized according to a uniform distribution over $[-w_{in}, w_{in}]$, where w_{in} is the maximum absolute value of an input weight. The sub-matrix organization of $\hat{\mathbf{W}}_{\mathcal{N}}$ corresponded to the case of undirected non-positional graphs, i.e. all the k sub-matrices of $\hat{\mathbf{W}}_{\mathcal{N}}$ were identical. Matrix $\hat{\mathbf{W}}_{\mathcal{N}}$ was initialized with values from a uniform distribution over $[-1, 1]$ and rescaled to the desired value of the contraction coefficient σ.

We considered GraphESNs with full connected reservoirs with 20, 30 and 50 units and a contraction coefficient σ varying from 0.5 to 2.1 [2] (with step 0.4). To initialize the input-to-reservoir matrix $\mathbf{W}_{in}$, we considered different values of w_{in}, namely 1.0, 0.1 and 0.01. For every setting of the hyper-parametrization we independently generated a number of 100 random (guessed) reservoirs of GraphESNs.

As both the datasets are related to structure-to-element transductions, we used a mean state mapping function to obtain a fixed-size state vector after the reservoir encoding process. For the sake of comparison with the GNN model, for the Mutagenesis dataset we considered also a supersource state mapping in which the supersource is arbitrarily selected as the first vertex in the AB description, as in [13]. However, note that generally using a mean state mapping function represents a choice of more general applicability than the arbitrary selection of one vertex to represent the whole input structure as in [13].

The readout was trained using linear regression, where $\mathbf{W}_{out}$ was computed by solving equation 11 using pseudo-inversion. Our code was written in C++, using the IT++4.0.6 library [3]. Pseudo-inversion was implemented through singular value decomposition, by using the IT++ function `svd`. The encoding process was terminated when the Euclidean distance between two consecutive global states was smaller than $\epsilon = 10^{-15}$.

The accuracy of the model was evaluated using cross-fold validation, with a number of 10 and 5 folds for Mutagenesis and PTC datasets, respectively, for the sake of comparison with results in [13], [16]. For hyper-parameters (model) selection, each training fold was split into a training and a validation set, with the selection of the hyper-parametrization

[2]Note that contractivity of the global state transition function $\hat{\tau}$ may hold in any norm. Therefore, values of the contraction coefficient σ not much greater than one, even if violating the condition on the Euclidean norm of equation 7, can still result in a convergent encoding process.

[3]http://itpp.sourceforge.net/current/index.html.

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on March 13,2021 at 05:31:35 UTC from IEEE Xplore. Restrictions apply.

TABLE I
AVERAGED AND BEST TEST ACCURACY (AND STANDARD DEVIATION) OF GRAPHESN WITH MEAN STATE MAPPING ON MUTAG.

	AB	AB+C	AB+C+PS
Average TS	76% (±9%)	80% (±6%)	80%(±6%)
Best TS	86% (±7%)	88% (±8%)	87% (±6%)

TABLE II
AVERAGED AND BEST TEST ACCURACY (AND STANDARD DEVIATION) OF GRAPHESN WITH SUPERSOURCE STATE MAPPING ON MUTAG.

	AB	AB+C	AB+C+PS
Average TS	72% (±4%)	82% (±7%)	82% (±7%)
Best TS	81% (±3%)	89% (±7%)	88% (±8%)

corresponding to the best validation accuracy, averaged over the 100 random guesses. As no further optimization was implemented, the best results over the selected 100 guesses are reported merely as an upper bound and an indication of the variance among the possible behaviors of the contractive fixed encoding model with linear readout. As obtained by using random guessing on the weight values of the fixed reservoir, such results might be interesting also as upper bounds for possible adaptive encoding approaches constrained by contractivity.

B. Results

The averaged and best (percentual) accuracies of GraphESNs on the Mutagenesis dataset with mean state mapping and supersource state mapping are reported in Table I and Table II, respectively, for the three possible descriptions of the compounds. With AB description only, the averaged accuracy of GraphESNs using mean state mapping is superior to the one obtained using supersource state mapping. For the other descriptions, AB+C and AB+C+PS, the averaged accuracies are very close, with supersource state mapping yielding a slightly better result.

Table III shows the averaged accuracy on the test set achieved by GNN and a selection of ILP methods for structured domains presented in [13], [21], [22], [23]. Table III allows a direct preliminary comparison with the averaged accuracy of GNNs, for which GraphESNs represent an architectural baseline (hence the goal was not to outperform the state-of-the-art results). Note that the accuracy of GNNs is available in [13] only for the full description AB+C+PS.

TABLE III
AVERAGED TEST ACCURACIES OF GNNS AND ILP METHODS ON MUTAG.

Model	AB	AB+C	AB+C+PS
GNN			90%
$1nn(d_m)$	81%	88%	
TILDE	77%	82%	
RDBC	83%	82%	

TABLE IV
AVERAGED AND BEST TEST ACCURACY (AND STANDARD DEVIATION) OF GRAPHESN WITH MEAN STATE MAPPING ON PTC

	MR	FR	MM	FM
Average TS	57%	65%	67%	58%
	(± 4%)	(± 3%)	(± 5%)	(± 4%)
Best TS	63%	70%	73%	65%
	(± 4%)	(± 2%)	(± 5%)	(± 5%)

TABLE V
AVERAGED TEST ACCURACY (AND STANDARD DEVIATION) OF KERNEL-BASED METHODS ON PTC

Method	MR	FR	MM	FM
MG-Kernel	63%	70%	69.0%	65%
	(±1%)	(±1%)	(±1%)	(±1%)
OA-Kernel	63%	70%	68%	65%
	(±2%)	(±1%)	(±2%)	(±1%)
EM-Kernel	61%	69%	67%	65%
	(±2%)	(±1%)	(±1%)	(±1%)

The averaged accuracy of GraphESNs is inferior to the accuracy of GNNs, which to our knowledge is the best result in literature on this dataset. It is also worth noting that the accuracy of trained GNNs is very similar to the averaged best accuracy of GraphESNs, which is confirmed to represent a bound to the potential results of approaches based on contractive encoding processes [4]. However, the comparison should take into account some differences among the two approaches, including the different implementations of the output function (GNNs use MLPs, while GraphESNs use a single layer of linear units) and the different nature of the information used (in GNNs different bond types in the AB description are distinguished by considering different edge labels in the corresponding graph representations). Moreover, we do not have sufficient details on the model selection used in [13] to guarantee that the two validation procedures are comparable. Therefore, a more complete comparison of the two models, considering also the other partial descriptions of the Mutagenesis dataset, is desirable. The accuracy of the GraphESN model, however, turns out to be even competitive with the results obtained by the ILP methods in Table III. As a general result, the variance of the GraphESNs reported in Tables I and II, effectively provides a reliable interval of the performance for models based on contractive dynamics.

Table IV reports the averaged and best test accuracy of GraphESNs on the four tasks of the PTC dataset. The accuracy on the same tasks obtained by three kernel-based methods for graphs domains, reported in [16] (marginalized (MG), Optimal Assignment and Expected Match kernels), is shown in Table V. Results of GraphESNs are loosely comparable with those of the kernel methods, also considering that the standard deviation seems relevant with respect to the

[4]Note that the cost of running the 100 GraphESN guesses is much less than the cost of thousands epochs in the training of GNN [13].

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on March 13,2021 at 05:31:35 UTC from IEEE Xplore. Restrictions apply.

differences among the accuracies. Even though the averaged accuracy of GraphESNs is inferior, the overlapping among the ranges of the accuracies seems significant, in particular for the MM and the FR tasks. Results of GraphESNs on the PTC dataset are also coherent with those obtained by other kernel-based methods on the same dataset (e.g. [17], [18]), although obtained using different validation procedures.

For what concerns the aspects related to the computational cost of GraphESNs, in our experiments, the encoding process required a number of steps within 6 an 13 for the Mutagenesis dataset and within 6 and 11 for the PTC dataset. On a laptop PC with a 1.73 GHz Pentium dual-core processor T2080, training and testing a GraphESN required less than 4 seconds and less than 5 seconds for the Mutagenesis and the PTC datasets, respectively.

V. Conclusions

We have presented a generalization of RC to structured domains processing, named the GraphESN. Exploiting the fixed contractive state dynamics typical of ESN models, convergence of the encoding is ensured for a large class of structured data including cyclic and undirected graphs, which entails mutual dependencies among state variables in the recursive approaches.

Efficiency is one of the key characteristics of the proposed model, as learning is restricted to a readout layer of feed-forward linear units. The encoding of the GraphESN has a computational cost that linearly scales with the dimension of the input graphs. As an example, for the sake of comparison, the cost of the encoding process of a GraphESN in the training phase is just equivalent to the cost of the encoding process in the test phase of a GNN [13].

Through experiments we have shown that even in the absence of training of recurrent connections, GraphESN is inherently able to discriminate among different graph structures. On two benchmark graph datasets we have shown that the performance achieved by state-of-the-art (more sophisticated) approaches are substantially within the range of variance of GraphESN results. Hence, such inherent prior discriminative capability represents a significant baseline for recursive models, especially for models based on contractive state dynamics, and we could empirically confirm the relevance of the assumption of contractivity over other possible architectural aspects. As for standard ESNs, the encoding capability can be even sufficient to achieve good predictive results according to the matching between the Markovian characterization of the state dynamics and the task at hand. This is therefore an interesting aspect for further investigations on the GraphESN model considering the extension of the Markovian characterization to graph topologies.

Though these issues certainly deserve further research, we believe that already in the present form the GraphESN, especially because of its efficiency and simplicity, can be a useful practical and analytical tool for complex relational applications. For instance, following the line of the presented applications, the emerging challenges in the toxicity field, concerning the aim of a first screening in the analysis of toxicity of potentially large collections of chemical compounds, can benefit from new efficient approaches for graphs.

References

[1] D. Verstraeten, B. Schrauwen, M. D'Haene, and D. Stroobandt, "An experimental unification of reservoir computing methods," *Neural Netw.*, vol. 20, no. 3, pp. 391–403, 2007.

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

[3] H. Jaeger and H. Haas, "Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication," *Science*, vol. 304, no. 5667, pp. 78–80, 2004.

[4] B. Hammer and P. Tiňo, "Recurrent neural networks with small weights implement definite memory machines," *Neural Computation*, vol. 15, no. 8, pp. 1897–1929, 2003.

[5] P. Tiňo, M. Cernanský, and L. Benusková, "Markovian architectural bias of recurrent neural networks," *IEEE Trans. Neural Netw.*, vol. 15, no. 1, pp. 6–15, 2004.

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

[7] A. Sperduti and A. Starita, "Supervised neural networks for the classification of structures," *IEEE Trans. Neural Netw.*, vol. 8, no. 3, pp. 714–735, 1997.

[8] P. Frasconi, M. Gori, and A. Sperduti, "A general framework for adaptive processing of data structures," *IEEE Trans. Neural Netw.*, vol. 9, no. 5, pp. 768–786, 1998.

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

[10] C. Gallicchio and A. Micheli, "TreeESN: a preliminary experimental analysis," accepted for ESANN 2010.

[11] B. Hammer, P. Tiňo, and A. Micheli, "A mathematical characterization of the architectural bias of recursive models," Universitat Osnabruck, Germany, Tech. Rep. 252, 2004.

[12] A. Micheli, "Neural network for graphs: A contextual constructive approach," *IEEE Trans. Neural Netw.*, vol. 20, no. 3, pp. 498–511, 2009.

[13] F. Scarselli, M. Gori, A.C.T., M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 61–80, 2009.

[14] M. Martelli, *Introduction to Discrete Dynamical Systems and Chaos*. Wiley, 1999.

[15] C. Gallicchio and A. Micheli, "Architectural and markovian factors of echo state networks," submitted to journal. Tech. Rep. TR-09-22 (Nov. 2009) http://compass2.di.unipi.it/TR/Files/TR-09-22.pdf.gz.

[16] H. Fröhlich, J. Wegner, and A. Zell, "Assignment kernels for chemical compounds," in *International Joint Conference on Neural Networks 2005 (IJCNN'05)*, 2005, pp. 913–918.

[17] H. Fröhlich, J. Wegner, F. Sieker, and A. Zell, "Optimal assignment kernels for attributed molecular graphs," in *ICML '05: Proc. of the 22nd Intern. Conf. on Machine Learning*. ACM, 2005, pp. 225–232.

[18] L. Ralaivola, S. J. Swamidass, H. Saigo, and P. Baldi, "Graph kernels for chemical informatics," *Neural Netw.*, vol. 18, no. 8, pp. 1093–1110, 2005.

[19] A. Srinivasan, S. M. R. King, and M. Sternberg, "Mutagenesis: ILP experiments in a non-determinate biological domain," in *Proc. of the 4th Int. Workshop Inductive Logic Programm.*, S. Wrobel, Ed., vol. 237, 1994, pp. 217–232.

[20] C. Helma, R. King, and S. Kramer, "The predictive toxicology challenge 2000-2001," in *Bioinformatics*, no. 17, 2001, pp. 107–108.

[21] J. Ramon, "Clustering and instance based learning in first order logic," PhD Dissertation, Dept. Comput. Sci., K.U. Leuven, Belgium, 2002.

[22] M. Kirsten, "Multirelational distance-based clustering," PhD Dissertation, Schl. Comput. Sci., Otto-von-Guericke Univ., Germany, 2002.

[23] L. D. Raedt and H. Blockeel, "Using logical decision trees for clustering," in *ILP '97: Proc. of the 7th International Workshop on Inductive Logic Programming*. London, UK: Springer-Verlag, 1997, pp. 133–140.

Authorized licensed use limited to: SUN YAT-SEN UNIVERSITY. Downloaded on March 13,2021 at 05:31:35 UTC from IEEE Xplore. Restrictions apply.