Ring Reservoir Neural Networks for Graphs

Claudio Gallicchio
Department of Computer Science
University of Pisa
Pisa, Italy
gallicch@di.unipi.it

Alessio Micheli
Department of Computer Science
University of Pisa
Pisa, Italy
micheli@di.unipi.it

***Abstract*—Machine Learning for graphs is nowadays a research topic of consolidated relevance. Common approaches in the field typically resort to complex deep neural network architectures and demanding training algorithms, highlighting the need for more efficient solutions. The class of Reservoir Computing (RC) models can play an important role in this context, enabling to develop fruitful graph embeddings through untrained recursive architectures. In this paper, we study progressive simplifications to the design strategy of RC neural networks for graphs. Our core proposal is based on shaping the organization of the hidden neurons to follow a ring topology. Experimental results on graph classification tasks indicate that ring-reservoirs architectures enable particularly effective network configurations, showing consistent advantages in terms of predictive performance.**

***Index Terms*—Graph Neural Networks, Graph Classification, Reservoir Computing, Reservoir Topology**

I. Introduction

Graph processing is assuming a central role in the development of the basic Machine Learning (ML) research [1]. Indeed, the possibility to consider the relationships among the singular samples in form of graphs and networks allows to overcome the limitations of models for flat domains, i.e. with fixed-size sample representations (fixed-length vectors) for the input data. As such, the direct treatment of this kind of data offers a remarkable opportunity to extend the possibility of successful application domains in fields that range from the processing of language structures or molecular data to the analysis of social or biological networks (just to mention some noteworthy examples).

The are many trends in the area of processing graph data in ML with a historical root in models for the adaptive processing for hierarchical data (rooted trees) [2]–[4], with pioneering applications to the Cheminformatics domains [5], which have been progressively extended to a family of models (and studies) based on the recursive approach. This family includes a wide range of approaches, from the unsupervised [6] and generative [7] areas, up to the extension to directed acyclic graphs [8]. All this models have been characterized by a recursive definition of the states (embedding) associated to each node of the input structures that naturally extends the state transition system of a recurrent neural network (RNN) for sequential data.

The direct processing of general directed/undirected and cyclic/acyclic graphs have been introduced following two basic approaches. The first one is based on a direct extension of the recursive neural networks, introduced by the Graph Neural Network (GNN) [9] and then for the Reservoir Computing framework by the Graph Echo State Network (GESN) [10]. Such approach uses a recursive model where the cyclic dependencies among states of the recursive transition system are allowed and treated by imposing constraints on the resulting dynamical system (resorting to a contractive dynamics to assure the convergence to a fixed-point representation for the state of each vertex of a graph). As a result, the fixed point of the recursive/dynamical system is exploited to represent (or *embed*) the input graphs. Once the states have been computed for all graph vertices, iterating the state transition function until convergence, the graph embedding values can be projected to the model output (readout), which is implemented as a standard layer of trained neural units. The diffusion of the context for each node of the graph is also guided by the iterations used to reach the stable point of the state values.

Differently from the recursive models, the other (coeval) line for graphs is to exploit the idea of stacking feed-forward layers of neural units to manage the mutual dependencies among state values that can occur in cyclic and/or undirected graphs. The multi-layer construction allows the context information to be diffused for the state of each input node in a compositional way. This line of research, initiated by the Neural Network for Graph (NN4G) [11], has been further developed under the general term of convolutional neural network (CNN) for graphs by many authors after the 2015, e.g. [12]–[16].

Finally, a further class of approaches that deserved to be mentioned (and that we will consider in our experimental part) is given by kernel methods for graphs [17]–[21].

Since the processing of complex types of data comes at the cost of a high computational demand, another trend concerns the efficiency issue. For the case of sequences and trees, the Reservoir Computing (RC) paradigm [22], in particular in the form of the so-called Echo State Network (ESN) [23], provides an approach for the efficient modeling of recurrent/recursive models [24]. Such paradigm is based on the use of fixed (randomized) values of the recurrent weights under stability conditions of the dynamical system (Echo State Property - ESP) [23], while the output units (readout) are the only trained part of the model. This line has been extended both to graph processing (i.e. the already mentioned GESN) and to deep architectures for trees [25] and graphs [26].

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

In this paper we follow the lines for efficient processing through RC approaches, with the aim to investigate on the effect of the neural network topology for processing graphs data. In particular, the proposed approach exploits the study on "minimum complexity" ESN for sequential data introduced in [27], in which the connections between the recurrent neurons of architecture follow a specific (ring shaped) topology and the stochastic elements (randomization in the recurrent weights initialization) are progressively eliminated, acting in accordance with specific deterministic rules. As a result, each instance of the model is fully described only by the value of a few hyper-parameters. The extension to graph domains lead us to introduce a new approach for graph processing by RC neural networks based on ring architectures.

Beside the investigation end, our proposal aims to introduce practical advantages in terms of a rational and simplified setting of the GESN, allowing also the user to increase the number of trials during the model cross-validation phase without increasing the computation time, or, in other words, to extend the exploration of the hyper-parameters space, which, in turns, increases the possibility to refine the application performance.

The rest of this paper is structured as follows. In Section II we give the basics of RC for graph processing. Then, in Section III we introduce the new proposed architectures, while in Section IV we analyze the results on different benchmarks at the state-of-the-art in the field of neural networks for graphs, also analyzing the effect of the architecture with respect to the number of units and the computation resource. Finally, in Section V we draw our conclusions.

II. Reservoir Computing for Graphs

Preliminaries on graphs. In this paper we focus on the problem of classifying undirected graph structures. A graph g is represented by the couple $g = (V_g, E_g)$, where V_g denotes the set of vertices and E_g is the set of edges. The number of vertices in g is indicated by N_g. The connectivity structure among the vertices of g is compactly represented by its adjacency matrix $\mathbf{A}_g$, which is a square (in our case symmetric) $N_g \times N_g$ matrix whose (i, j) element is always 0 unless there is an edge connecting vertex i to vertex j. The neighborhood of each vertex $v \in V_g$ is the set of vertices that are adjacent to v, denoted as $\mathcal{N}(v) = \{v' \in \mathbf{V}_g | (v, v') \in E_g\}$. The maximum among the sizes of the neighborhoods defines the degree k. We consider vertex-labelled graphs, where an input feature vector is assigned to each vertex. Here we use the notation $\mathbf{u}(v)$ to indicate the input feature vector attached to vertex v. To ease the notation, in the the rest of the paper we drop the subscript g whenever the reference to the graph in question is unambiguous.

Graph Echo State Networks. Reservoir Computing (RC) [22] is a popular design paradigm for efficiently trained dynamical neural models, extended to process graph structures with the introduction of the Graph Echo State Network (GESN) in [10].

Architecturally, a GESN is composed by a hidden layer of recursive non-linear neurons, the *reservoir*, that implements the graph embedding process, followed by a simple linear feed-forward readout layer that computes the output. Crucially, and differently from conventional recurrent graph neural network approaches [9], the weights on the hidden (reservoir) layer's connections are left untrained after initialization, and only the parameters of the readout are subject to training.

We use N_I, N_H and N_O to respectively indicate the number of input, hidden (i.e., reservoir) and readout units. The reservoir's neurons encode each vertex v of an input graph into a state representation (i.e., an internal neural embedding) by means of a state transition function defined as follows[1]:

$$\mathbf{x}(v) = \tanh\left(\mathbf{V}\,\mathbf{u}(v) + \sum_{v' \in \mathcal{N}(v)} \mathbf{W}\,\mathbf{x}(v')\right), \tag{1}$$

where $\mathbf{x}(v) \in \mathbb{R}^{N_R}$ is the state computed for vertex v, $\mathbf{u}(v) \in \mathbb{R}^{N_I}$ is the the input feature vector attached to vertex v, $\mathbf{V} \in \mathbb{R}^{N_H \times N_I}$ is the input weight matrix, $\mathbf{W} \in \mathbb{R}^{N_H \times N_H}$ is the reservoir recurrent weight matrix, and $\tanh$ indicates the element-wise application of the hyperbolic tangent non-linearity. As described by (1), the state for each vertex v depends on both its input features, modulated by the weights in $\mathbf{V}$, and on the set of states computed for the vertices in the neighborhood of v, modulated by the weights in $\mathbf{W}$. In line with RC for time-series, the reservoir neurons are sparsely and randomly connected among each other. This implies that $\mathbf{W}$ is instantiated as a sparse matrix, hence speeding-up the state transition computation, which scales linearly with the number of reservoir neurons, with the degree and with the number of vertices [26]. The reservoir recurrent structure and the state transition computation (applied to a vertex in the input graph) are graphically illustrated in Fig. 1.

The operation of the reservoir on an entire input graph can be collectively represented in the (more compact) form:

$$\mathbf{X} = \tanh(\mathbf{V}\mathbf{U} + \mathbf{W}\mathbf{X}\mathbf{A}), \tag{2}$$

where $\mathbf{U} \in \mathbb{R}^{N_I \times N}$ and $\mathbf{X} \in \mathbb{R}^{N_H \times N}$ column-wise collect respectively the input feature vectors and states for all the N vertices in the given input graph, and $\mathbf{A}$ is the adjacency matrix. Interestingly, (2) can be studied as the function driving the dynamics of an input-driven non-linear dynamical system, where the role of exciting (external) input information is played by both $\mathbf{U}$ and $\mathbf{A}$ (that depend on the graph to which the reservoir system is applied). Existence and uniqueness of solutions of (2), i.e. of the graph embedding computed by the reservoir, can be guaranteed by imposing a stability property named *Graph Embedding Stability* (GES) [26]. A necessary condition for the GES consists in scaling the effective spectral radius, i.e. $\rho = \rho(\mathbf{W})\,k$, to a value smaller than 1, where k denotes the degree of the set of graphs under consideration and $\rho(\mathbf{W})$ is the largest eigenvalue of $\mathbf{W}$ in modulus [26]. The reservoir parameters in matrix $\mathbf{W}$ can then be randomly initialized from a uniform distribution over $[-1, 1]$, and then re-scaled to meet the $\rho < 1$ condition. Analogously, weights in the input matrix $\mathbf{V}$ are randomly

[1] We drop the bias terms in the equations for the ease of notation.

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

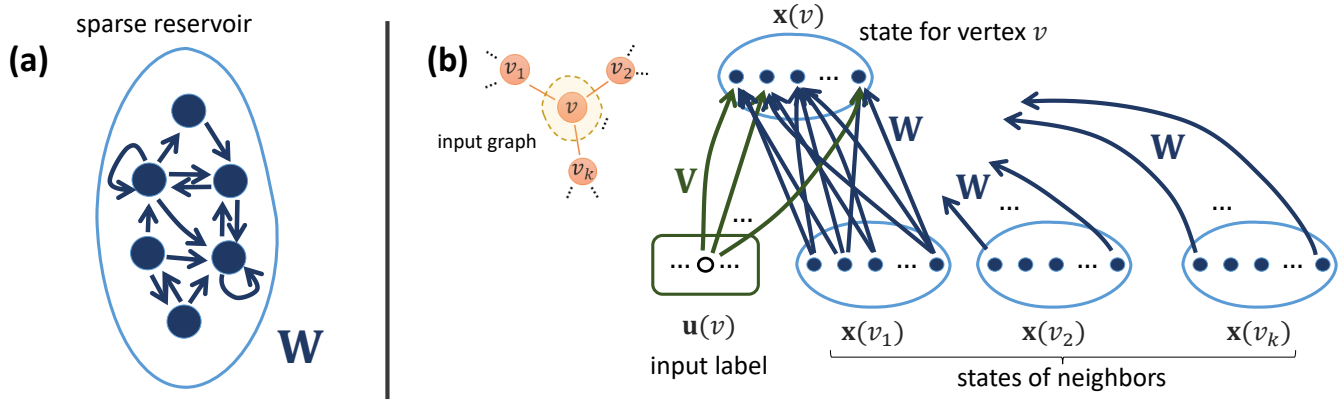

Fig. 1. GESN: sparse reservoir recurrent structure (a), and its application to graph encoding (b).

initialized from a uniform distribution over $[-\omega, \omega]$, where ω is an input-scaling parameter that determines the strength of the input influence in driving the reservoir dynamics. The values of ρ and ω are treated as hyper-parameters. Given an input graph, (2) is then iterated until convergence[2] to the unique fixed point of the corresponding dynamical system. Such a fixed point in the reservoir state space depends on the specific input graph (through matrices $\mathbf{U}$ and $\mathbf{A}$) and represents its developed (neural) embedding. In practice, the dynamical graph embedding process is stopped whenever the Euclidean distance between the states in successive iterations of (2) goes below a threshold ε, or a maximum number of iterations ν has been reached. After initialization, the weights in both $\mathbf{V}$ and $\mathbf{W}$ are left untrained, hence the striking efficiency advantage with respect to the approach in [9], where stability is not imposed by construction at initialization, but rather obtained by a possibly long and costly constrained training process.

The output computation is performed by a readout layer, that linearly combines the representations developed for each vertex of an input graph. The readout computation can be described as follows:

$$\mathbf{y}(g) = \mathbf{W}_o \sum_{v \in V_g} \mathbf{x}(v), \tag{3}$$

where $\mathbf{W}_o \in \mathbb{R}^{N_O \times N_H}$ is the readout weight matrix that is adjusted on a set of training samples. Note that the readout operation in (2) includes the application of a global sum-pooling (or aggregation) operator before the linear combination. Training of $\mathbf{W}_o$ is performed in closed-form using Tikhonov regularization as in classical RC approaches [22].

III. Ring Reservoirs for Graphs

Given the untrained and randomized approach to the design of network's dynamics, a commonly arising question in the RC field is how to create "better" reservoirs than just random ones. To keep the computational advantage of untrained recurrent connections, a possible way to address this question consists in trying to optimize the architecture of the reservoir beforehand. One interesting concept emerged in the literature on dynamical neural models for time-series processing is orthogonality of recurrence matrices. In the temporal domain, this kind of recurrence systems are known to be naturally biased towards having improved memorization skills [28]. Although several ways exist to "orthogonalize" a dynamical neural model (see, e.g., [29], [30]), a particularly effective approach is to impose a ring-constrained pattern of connectivity among the recurrent neurons in order to connect them to form a cycle. A number of studies in the context of time-series processing highlighted several advantages of such ring constrained topologies, e.g. in terms of enriched quality of developed representations [31], improved memory and predictive performance [27], [30]. For the first time in literature, in this paper we investigate the effects of this type of constrained architectures in the field of neural networks for graph processing. We introduce the following two new models that progressively simplify the construction of RC networks for graphs.

Graph Ring-Reservoir Network (GRN) – Connections among reservoir neurons are organized according to a ring topology, where each neuron propagates its activation to the successive one and is fed by the previous one in the cycle. The resulting reservoir weight matrix $\mathbf{W}$ has the shape of a permutation matrix $\mathbf{P}$, where all entries are 0, except on the subdiagonal and on the top-right corner, i.e.:

$$\mathbf{P} = \begin{bmatrix} 0 & 0 & \dots & 1 \\ 1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 1 & 0 \end{bmatrix}, \quad \mathbf{W} = \lambda\, \mathbf{P}. \tag{4}$$

All the non-zero weights in $\mathbf{W}$ are set to the same value of λ, which directly controls $\rho(\mathbf{W})$ and hence can be used to tune the effective spectral radius of the reservoir system through the relation $\rho = \lambda\, k$. The resulting state transition function in (2) is simplified as follows:

$$\mathbf{X} = \tanh(\mathbf{V}\,\mathbf{U} + \lambda\,\mathbf{P}\mathbf{X}\mathbf{A}). \tag{5}$$

[2]Typically, the reservoir is initialized to a zero state for each vertex in the graph, although any initial condition in the state space would be just as good due to the uniqueness of the solution of (2).

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

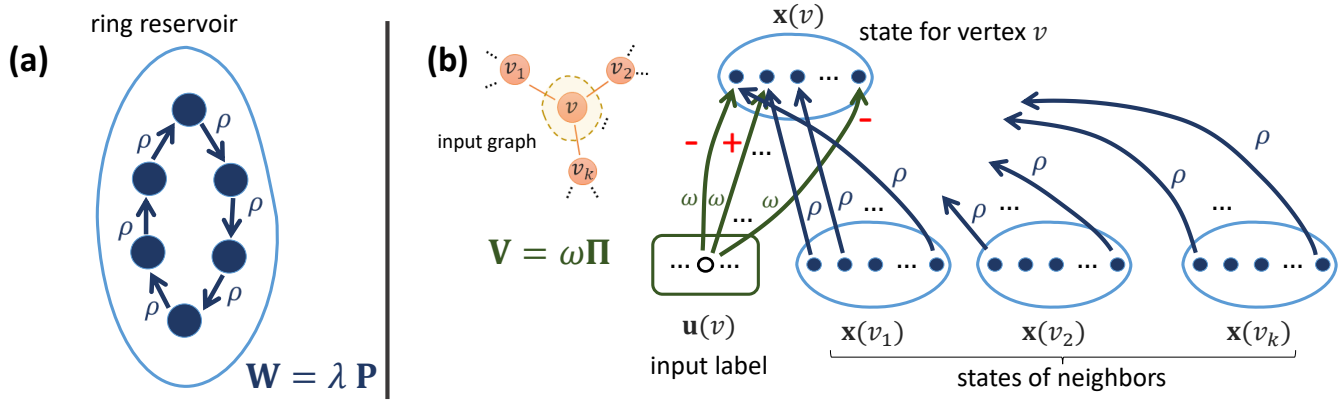

Fig. 2. MGN: ring-shaped topology of the reservoir (a), and its application to graph encoding (b). The same reservoir topology is used in GRN.

Notice that the effects of stochasticity in the construction of the reservoir are partially removed. The recurrent reservoir weights are indeed set in a deterministic way, and the content of matrix $\mathbf{W}$ is fully described by just one number (i.e., the spectral radius). Randomization enters the network's design solely as regards the setup of the input weight matrix $\mathbf{V}$.

Minimal Graph Network (MGN) – The reservoir weight matrix $\mathbf{W}$ is ring-shaped as in the case of GRN. In addition, we impose an architectural simplification also to the input weight matrix $\mathbf{V}$. Specifically, all the input weights are set to the same absolute value ω, while the signs of the weights are chosen deterministically similarly to [27], which introduced the idea of "minimal complexity" in recurrent architectures. We construct a sign weight matrix $\mathbf{\Pi}$ of the same size as $\mathbf{V}$. Following the (aperiodic) decimal expansion of the irrational number π, we row-wise generate the entries in $\mathbf{\Pi}$ using the value -1 if the corresponding digit of π is smaller than 5, and the value $+1$ otherwise. Accordingly, we have that $\mathbf{V} = \omega\,\mathbf{\Pi}$, and the reservoir state transition function in (2) is further simplified as follows:

$$\mathbf{X} = \tanh(\omega\,\mathbf{\Pi}\,\mathbf{U} + \lambda\,\mathbf{P}\mathbf{X}\mathbf{A}). \tag{6}$$

Fig. 2 illustrates the simplified reservoir architecture and the state computation in MGN networks. Note that the same ring reservoir topology is used also for GRN.

Interestingly, the reservoir of MGN is constructed in a fully deterministic fashion, and any aspect of randomization in its setup has been eliminated. The reservoir system is now completely described by just 2 numbers, i.e., the input scaling parameter ω and the spectral radius ρ. The network initialization process is thereby greatly simplified, reducing its degrees of freedom - in comparison to the case of GESN - by a factor that scales with the reservoir size N_H.

For both GRN and MGN, the readout operation and training is the same as already described for GESN in Section II.

IV. Experiments

We perform an experimental evaluation of the proposed GRN and MGN models, by assessing their predictive performance on several graph classification benchmarks. Our analysis is performed comparatively to both GESN (through our experiments) and state-of-the-art neural networks and kernel methods for graphs (through literature results).

Datasets. We take into consideration 6 graph classification datasets (publicly available online [32]). Two of them, MUTAG [33] and NCI1 [34], come from the cheminformatics domain, where each input graph is used as representation of a chemical compound: each vertex stands for an atom of the molecule, and edges between vertices represent bonds between atoms. In the case of MUTAG, the dataset contains nitroaromatic compounds and the target classification represents mutagenicity on Salmonella typhimurium. In the case of NCI1, the dataset is relative to anti-cancer screens where the chemicals are assessed as positive or negative to cell lung cancer. For both MUTAG and NCI1, each vertex has an input feature vector representing the corresponding atom type, encoded by a one-hot-encoding scheme into a vector of 0/1 elements. The other datasets that we use come from the social network analysis domain and are introduced in [18]. Two of them, i.e., IMDB-BINARY (IMDB-2) and IMDB-MULTI (IMDB-m), are datasets of movie collaboration, where each graph represents the ego network of an actor/actress and the target classification pertains the movie genre (2 for IMDB-2, 3 for IMDB-m). Then, we consider the REDDIT (binary) dataset, where each input graph represents an online discussion thread to be classified in one of 2 possible types of discussions. The last dataset that we use is COLLAB, a collection of graphs representing the ego-networks of researchers, classified according to their areas of research. For all the social network benchmarks, there is no label attached to each vertex in the input graphs, and we use a fixed (uni-dimensional) 1 value as input feature for each vertex. For all binary classification problems we encoded the target class as a value in $\{-1, +1\}$. For multi-class classification benchmarks each target is represented through a -1/+1 one-hot-encoding of the corresponding class. Table I shows a summary of datasets information.

Experimental Settings. We conducted experiments with GRN, MGN and GESN. For all the considered RC networks we used dense input weight matrices $\mathbf{V}$ and sparse reservoir

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

Dataset	# graphs	# vertices (tot)	# vertices (avg)	# classes
MUTAG	188	3371	17.9	2
IMDB-b	1000	19773	19.8	2
IMDB-m	1500	19502	13.0	3
REDDIT	2000	859254	429.6	2
NCI1	4110	122747	29.9	2
COLLAB	5000	372474	74.5	3

TABLE I
SUMMARY OF DATASETS INFORMATION.

weight matrices $\mathbf{W}$. For a fair comparison, in the case of GESN we used a sparse pattern of connectivity such that each reservoir neuron is connected to only 1 other reservoir neuron. In this way, the degree of reservoir sparsity is comparable in all the considered settings, with the crucial difference that in GRN and MGN such sparsity is not randomized but rather structured following the ring topology described in Section III. As regards the stop conditions for the dynamical graph embedding process, we used a convergence threshold of $\varepsilon = 10^{-3}$, and a maximum number of iterations $\nu = 50$. The hyper-parameters ω and ρ were explored by random search, i.e., generating $\mathcal{C} = 50$ random reservoir configurations where ω and ρ were sampled from a uniform distribution on $(0, 1)$. For every reservoir configuration, the Tikhonov regularizer for the readout training was explored in a log-scale grid $\{10^{-10}, 10^{-9}, \ldots, 10^{5}\}$. We ran experiments considering progressively larger reservoirs, with N_H in the range $\{5, 10, 30, 50\}$ for MUTAG, IMDB-2, IMDB-m, and REDDIT, and in the range $\{50, 100, 300, 500\}$ for NCI1 and COLLAB.

To account for randomization in the initialization of the reservoir, for each of the $\mathcal{C}$ reservoir configurations we instantiated a number of $\mathcal{R} = 50$ repetitions (i.e., reservoir guesses), averaging the error on such repetitions. The total number of generated networks is then equal to $\mathcal{C} \times \mathcal{R}$. Note that for MGN the reservoir initialization is fully deterministic and there is no need to average the performance on multiple repetitions. We thereby considered two possible experimental settings for MGN hyper-parameters search: one "complete" and one "reduced". In the former we increase the number of explored MGN configurations such that the total number of generated networks is the same as for GRN and GESN, while in the latter we explore the same number of configurations as for GRN and GESN (hence, the number of generated MGN networks is smaller). More specifically, in the complete setting we put on par the total effort in the hyper-parameter search, and generated the same total number of reservoir networks as for the other models. In this case, the experiments with MGN span a total number of $\mathcal{C} \times \mathcal{R}$ possible configurations. In the reduced setting, we put on par the number of explored configurations, and limit ourselves to considering a total number of $\mathcal{C}$ MGN networks. In this case, the hyper-parameter search for MGN is $\mathcal{R}$ times faster than those for GRN and GESN.

The output class for binary classification tasks was computed by applying the readout equation (3), followed by the sign function to discretize the output in $\{-1, +1\}$. For multi-class classification tasks, for each graph the output class was assigned in correspondence of the readout unit with the highest activation. We computed the performance in terms of accuracy, following a stratified 10-fold cross validation scheme. The values of the hyper-parameters were chosen on each fold by model selection (individually for each model), using a nested level of stratified 10-fold cross validation.

Results. We first analyzed the behavior of the proposed approaches at the increase of the reservoir network size. To this end, we computed the performance achieved by each model, optimizing the hyper-parameters individually for each possible number of reservoir neurons. The results are shown in Fig. 3, in which we report the validation accuracy achieved by GRN and MGN (under both complete and reduced settings) in comparison to GESN. Fig. 3 indicates a clear general trend, with both GRN and MGN having higher accuracy than GESN consistently for all reservoir sizes on all tasks. In particular, the performance gap is evidently wider in the case of MGN. Interestingly, despite the architectural simplifications, minimal ring-shaped reservoirs show a higher fitting potential. Already under reduced search settings, MGN has higher performance than GESN and GRN (with almost on par results on COLLAB). Under complete search settings, MGN beats all other models, showing an effective enhancement in the exploration of the RC hyper-parameters space. Remarkably, on 4 over 6 tasks, the smallest MGN (complete) performs in line with, or better than, the largest (10 times bigger) GESN.

We then analyze the overall generalization performance of the proposed architectures by individually optimizing (through model selection) the hyper-parameters of each RC model, taking collectively into account all the possible reservoir sizes. The achieved test accuracies are given in Tab. II. In the same table, we report the performance of a significant selection of state-of-the-art ML models for graph classification from the literature. We considered the following graph neural networks: Fast and Deep Graph Neural Network (FDGNN) [26], a recently introduced deep RC approach for graphs, Deep Graph Convolutional Neural Network (DGCNN) [12], Parametric Graph Convolution DGCNN (PGC-DGCNN) [13], DiffPool [37], Edge-Conditioned Convolution network (ECC) [38], Graph Isomorphism Network (GIN) [16] GraphSAGE [39], Graph Neural Network [9] and Relational Neural Network [40]. We also considered a number of kernel methods for graphs: Graphlet Kernel (GK) [17], Deep GK (DGK), Weisfeiler-Lehman Kernel (WL) [21], and Propagation Kernel (PK) [20]. Results for these models are quoted (when available) from the literature references indicated in the table, where the experimental settings for model selection were as close as possible to be rigorous as ours (the only exception is the result of DGCNN on REDDIT, which is taken from [35]).

Tab. II confirms the already observed general trend of ring-constrained RC networks variants beating standard GESN. Specifically, both GRN and MGN (in each of the search settings) improve the performance of GESN on 5 out of 6 tasks. A remarkable case is given by the largest dataset, i.e., COLLAB, where the performance boost is particularly evident

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

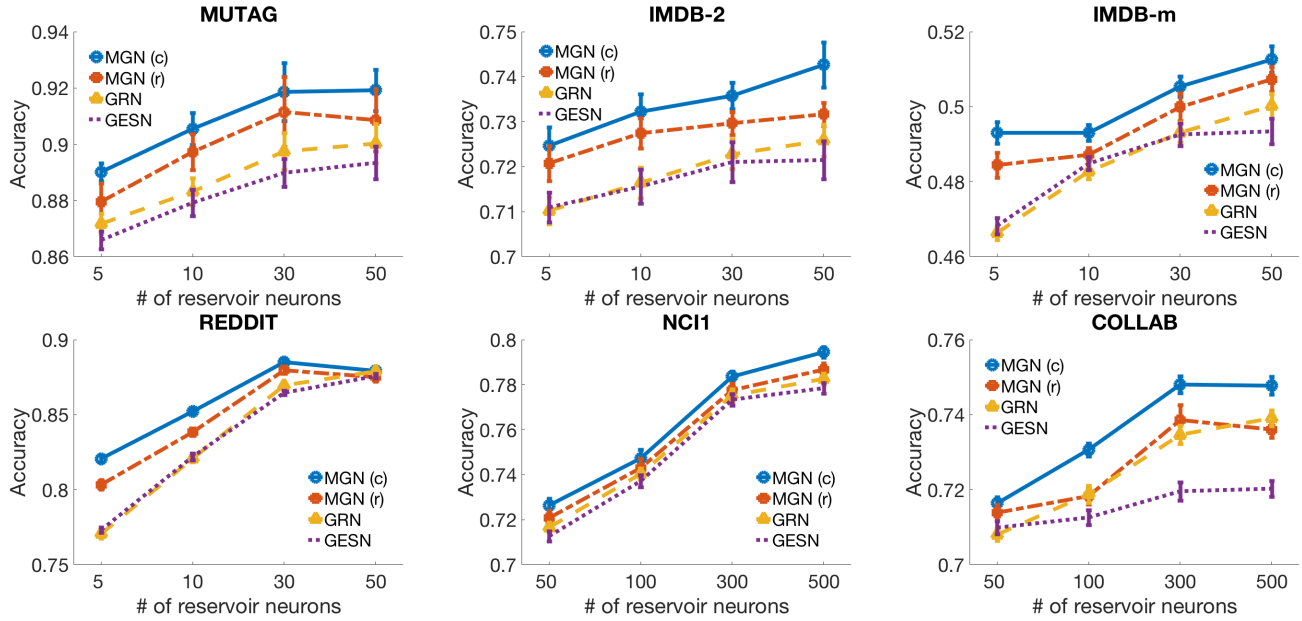

Fig. 3. Validation accuracy or RC models for graph classification for increasing reservoir size. Results are averaged (and std computed) on the outer 10 folds.

	MUTAG	IMDB-2	IMDB-m	REDDIT	NCI1	COLLAB
MGN (complete)	$87.8_{(\pm 6.3)}$	$\mathbf{72.7}_{(\pm 3.2)}$	$49.5_{(\pm 2.2)}$	$87.7_{(\pm 1.7)}$	$78.8_{(\pm 2.3)}$	$74.5_{(\pm 1.4)}$
MGN (reduced)	$88.3_{(\pm 5.6)}$	$72.0_{(\pm 3.6)}$	$50.2_{(\pm 2.0)}$	$88.1_{(\pm 1.8)}$	$77.6_{(\pm 2.4)}$	$74.1_{(\pm 1.3)}$
GRN	$88.4_{(\pm 7.6)}$	$71.7_{(\pm 2.8)}$	$\mathbf{50.5}_{(\pm 1.4)}$	$87.6_{(\pm 1.4)}$	$78.2_{(\pm 2.2)}$	$73.8_{(\pm 1.4)}$
GESN	$88.2_{(\pm 6.3)}$	$71.7_{(\pm 3.6)}$	$48.7_{(\pm 2.1)}$	$87.5_{(\pm 1.1)}$	$77.8_{(\pm 2.0)}$	$72.1_{(\pm 1.6)}$
FDGNN [26]	$\mathbf{88.5}_{(\pm 3.8)}$	$72.4_{(\pm 3.6)}$	$50.0_{(\pm 1.3)}$	$\mathbf{89.5}_{(\pm 2.2)}$	$77.8_{(\pm 1.6)}$	$74.4_{(\pm 1.0)}$
DGCNN [12]	$85.8_{(\pm 1.7)}$	$70.0_{(\pm 0.9)}$	$47.8_{(\pm 0.9)}$	$77.1_{(\pm 2.9)}$	$74.4_{(\pm 0.5)}$	$73.8_{(\pm 0.5)}$
PGC-DGCNN [13]	$87.2_{(\pm 1.4)}$	$71.6_{(\pm 1.2)}$	$47.3_{(\pm 1.4)}$	–	$76.1_{(\pm 0.7)}$	$75.0_{(\pm 0.6)}$
DiffPool [35]	–	$68.3_{(\pm 6.1)}$	$45.1_{(\pm 3.2)}$	$76.6_{(\pm 2.4)}$	$76.9_{(\pm 1.9)}$	$67.7_{(\pm 1.9)}$
ECC [35]	–	$67.8_{(\pm 4.8)}$	$44.8_{(\pm 3.1)}$	–	$76.2_{(\pm 1.4)}$	–
GIN [35]	–	$66.8_{(\pm 3.9)}$	$42.2_{(\pm 4.6)}$	$87.0_{(\pm 4.4)}$	$80.0_{(\pm 1.4)}$	$\mathbf{75.9}_{(\pm 1.9)}$
GraphSAGE [35]	–	$69.9_{(\pm 4.6)}$	$47.7_{(\pm 3.6)}$	$86.1_{(\pm 2.0)}$	$76.0_{(\pm 1.8)}$	$71.6_{(\pm 1.5)}$
GNN [36]	$80.5_{(\pm 0.8)}$	–	–	–	–	–
RelNN [36]	$87.8_{(\pm 2.5)}$	–	–	–	–	–
GK [18]	$81.4_{(\pm 1.7)}$	$65.9_{(\pm 1.0)}$	$43.9_{(\pm 0.4)}$	$77.3_{(\pm 0.2)}$	$62.5_{(\pm 0.3)}$	$72.8_{(\pm 0.6)}$
DGK [18]	$82.7_{(\pm 1.5)}$	$67.0_{(\pm 0.6)}$	$44.6_{(\pm 0.5)}$	$78.0_{(\pm 0.4)}$	$62.5_{(\pm 0.3)}$	$73.1_{(\pm 0.3)}$
WL [12]	$84.1_{(\pm 1.9)}$	–	–	–	$\mathbf{84.5}_{(\pm 0.5)}$	–
PK [12]	$76.0_{(\pm 2.7)}$	–	–	–	$82.5_{(\pm 0.5)}$	–

TABLE II

TEST ACCURACY OF RC MODELS FOR GRAPH CLASSIFICATION (COMPARED TO LITERATURE METHODS). RESULTS ARE AVERAGED (AND STD COMPUTED) ON THE OUTER 10 FOLDS. THE BEST OVERALL ACCURACY ON EACH TASK IS HIGHLIGHTED IN BOLD FONT.

(up to more than 2% of accuracy). We find it particularly intriguing that the observed accuracy improvement is obtained by just by imposing a specific (and rather simple) structure to the architectural setup of the network. In the case of MGN this architectural effect is combined with the already noted possibility to search the hyper-parameters space more extensively. Results of ring reservoir networks compare well also with literature results. In this regard, notice that while literature works on deep learning for graphs often imply complex end-to-end fully trained multi-layered architectures with several thousands of trainable weights, our results here were achieved with very simple (single-layered) recursive untrained architectures, and entail a maximum number of trained weights of 501 for NCI1 and COLLAB, and of 51 for all the other tasks. Despite this, our proposed models establish new sate-of-the-art results on IMDB-2 and IMDB-m. On MUTAG, REDDIT and COLLAB our achieved performances are very close to the highest ones in literature, while on NCI1 - where WL and PK kernels score best - the obtained accuracy is not far from the best performing neural network (GIN). Finally, looking also at the accuracy of FDGNN, we find it interesting to point out that on 4 out of 6 tasks, the highest accuracy is achieved by a model in the RC class (either in a shallow, or deep architectural setting).

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

V. Conclusions

We have studied simple RC architectures for graph processing where the hidden recurrent neurons are constrained to a ring topology. The introduced simplifications allowed us to progressively reduce, up to eliminate, the aspects of stochasticity in the initialization of reservoir networks for graphs. Our experimental analysis indicated that such "minimal" RC architectures enable an effective exploration of the hyper-parameters, often leading to an improved performance. The provided results also pointed out that despite their simplicity, ring-reservoir neural networks are particularly effective in graph classification benchmarks, reaching a performance that is comparable to (and sometimes even better than) that of several more complex ML models for graphs.

The evidences illustrated in this paper naturally support future research investigations, aiming at extending the observed advantages of constrained reservoir architectures both to the construction of more effective deep RC models and to the design of more efficient learning algorithms for trained neural networks for graphs.

References

[1] D. Bacciu, F. Errica, A. Micheli, and M. Podda, "A gentle introduction to deep learning for graphs," *arXiv preprint arXiv:1912.12693*, 2019.

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

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

[4] B. Hammer, *Learning with Recurrent Neural Networks*, ser. Springer Lecture Notes in Control and Information Sciences. Springer-Verlag, 2000, vol. 254.

[5] A. Bianucci, A. Micheli, A. Sperduti, and A. Starita, "Application of cascade correlation networks for structures to chemistry," *Applied Intelligence*, vol. 12, no. 1/2, pp. 117–146, 2000.

[6] B. Hammer, A. Micheli, A. Sperduti, and M. Strickert, "Recursive self-organizing network models," *Neural Networks*, vol. 17, no. 8-9, pp. 1061–1085, 2004.

[7] D. Bacciu, A. Micheli, and A. Sperduti, "Compositional generative mapping for tree-structured data - part I: Bottom-up probabilistic modeling of trees," *IEEE transactions on neural networks and learning systems*, vol. 23, no. 12, pp. 1987–2002, 2012.

[8] A. Micheli, D. Sona, and A. Sperduti, "Contextual processing of structured data by recursive cascade correlation," *IEEE Transactions on Neural Networks*, vol. 15, no. 6, pp. 1396–1410, 2004.

[9] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.

[10] C. Gallicchio and A. Micheli, "Graph echo state networks," in *The 2010 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2010, pp. 1–8.

[11] A. Micheli, "Neural network for graphs: A contextual constructive approach," *IEEE Transactions on Neural Networks*, vol. 20, no. 3, pp. 498–511, 2009.

[12] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[13] D. V. Tran, N. Navarin, and A. Sperduti, "On filter size in graph convolutional networks," in *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2018, pp. 1534–1541.

[14] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 1993–2001.

[15] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *International conference on machine learning*, 2016, pp. 2014–2023.

[16] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *Proceedings of ICLR 2019*, 2018.

[17] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt, "Efficient graphlet kernels for large graph comparison," in *Artificial Intelligence and Statistics*, 2009, pp. 488–495.

[18] P. Yanardag and S. Vishwanathan, "Deep graph kernels," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 1365–1374.

[19] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, "Graph kernels," *Journal of Machine Learning Research*, vol. 11, no. Apr, pp. 1201–1242, 2010.

[20] M. Neumann, R. Garnett, C. Bauckhage, and K. Kersting, "Propagation kernels: efficient graph kernels from propagated information," *Machine Learning*, vol. 102, no. 2, pp. 209–245, 2016.

[21] N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *Journal of Machine Learning Research*, vol. 12, no. Sep, pp. 2539–2561, 2011.

[22] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Computer Science Review*, vol. 3, no. 3, pp. 127–149, 2009.

[23] 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.

[24] C. Gallicchio and A. Micheli, "Tree echo state networks," *Neurocomputing*, vol. 101, pp. 319–337, 2013.

[25] ——, "Deep reservoir neural networks for trees," *Information Sciences*, vol. 480, pp. 174–193, 2019.

[26] ——, "Fast and deep graph neural networks," in *Proceedings of AAAI 2020*, 2020, arXiv preprint arXiv:1911.08941.

[27] A. Rodan and P. Tino, "Minimum complexity echo state network," *IEEE transactions on neural networks*, vol. 22, no. 1, pp. 131–144, 2010.

[28] O. L. White, D. D. Lee, and H. Sompolinsky, "Short-term memory in orthogonal neural networks," *Physical review letters*, vol. 92, no. 14, p. 148102, 2004.

[29] I. Farkaš, R. Bosák, and P. Gergel', "Computational analysis of memory capacity in echo state networks," *Neural Networks*, vol. 83, pp. 109–120, 2016.

[30] T. Strauss, W. Wustlich, and R. Labahn, "Design strategies for weight matrices of echo state networks," *Neural computation*, vol. 24, no. 12, pp. 3246–3276, 2012.

[31] P. Tino, "Dynamical systems as temporal feature spaces," *arXiv preprint arXiv:1907.06382*, 2019.

[32] K. Kersting, N. M. Kriege, C. Morris, P. Mutzel, and M. Neumann, "Benchmark data sets for graph kernels," 2016. [Online]. Available: http://graphkernels.cs.tu-dortmund.de

[33] A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch, "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity," *Journal of medicinal chemistry*, vol. 34, no. 2, pp. 786–797, 1991.

[34] N. Wale, I. A. Watson, and G. Karypis, "Comparison of descriptor spaces for chemical compound retrieval and classification," *Knowledge and Information Systems*, vol. 14, no. 3, pp. 347–375, 2008.

[35] F. Errica, M. Podda, D. Bacciu, and A. Micheli, "A fair comparison of graph neural networks for graph classification," in *International Conference on Learning Representations*, 2020.

[36] W. Uwents, G. Monfardini, H. Blockeel, M. Gori, and F. Scarselli, "Neural networks for relational learning: an experimental comparison," *Machine Learning*, vol. 82, no. 3, pp. 315–349, 2011.

[37] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Advances in Neural Information Processing Systems*, 2018, pp. 4800–4810.

[38] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 3693–3702.

[39] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in neural information processing systems*, 2017, pp. 1024–1034.

[40] H. Blockeel and M. Bruynooghe, "Aggregation versus selection bias, and relational neural networks," in *IJCAI-2003 Workshop on Learning Statistical Models from Relational Data, SRL-2003, Acapulco, Mexico*, 2003.

978-1-7281-6926-2/20/$31.00 ©2020 IEEE