Fast and Deep Graph Neural Networks

Claudio Gallicchio, Alessio Micheli
Department of Computer Science, University of Pisa
Largo B. Pontecorvo, 3. 56127 Pisa, Italy
{gallicch, micheli}@di.unipi.it

Abstract

We address the efficiency issue for the construction of a deep graph neural network (GNN). The approach exploits the idea of representing each input graph as a fixed point of a dynamical system (implemented through a recurrent neural network), and leverages a deep architectural organization of the recurrent units. Efficiency is gained by many aspects, including the use of small and very sparse networks, where the weights of the recurrent units are left untrained under the stability condition introduced in this work. This can be viewed as a way to study the intrinsic power of the architecture of a deep GNN, and also to provide insights for the set-up of more complex fully-trained models. Through experimental results, we show that even without training of the recurrent connections, the architecture of small deep GNN is surprisingly able to achieve or improve the state-of-the-art performance on a significant set of tasks in the field of graphs classification.

1 Introduction

Graphs are relevant data structures that provide an useful abstraction for many kind of real data, ranging from molecular data to social and biological networks (just to mention the most noteworthy cases), and for all the cases characterized by data with relationships. The direct treatment of this kind of data allows a Machine Learning (ML) system to consider the vector patterns, which characterize the standard flat domain, and the relationships among them, respecting in such a way the inherent nature of the underlying structured domain.

It is not surprising, then, that there is a long tradition of studies for the processing of structured data in ML, starting for the Neural Networks (NN) since the '90s, up to current increasing interest in the field of deep learning for graphs. Among the first NN approaches we can mention the Recursive Neural Network (RecNN) models for tree structured data in (Sperduti and Starita 1997; Frasconi, Gori, and Sperduti 1998), and more recently in (Socher et al. 2011; 2013), which have been progressively extended to directed acyclic graph (Micheli, Sona, and Sperduti 2004). Such approaches provided a neural implementation of a state transition system traversing the input structures in order to make the embedding and subsequently the classification of the input hierarchical data. The main issue in extending such approaches to general graphs (cyclic/acyclic, directed/undirected) was the processing of cycles due to the mutual dependencies occurring among the state variables definitions represented in the neural recursive units. The earliest approaches for graphs were the Graph Neural Network NN (GNN)(Scarselli et al. 2009) and the Neural Network for Graphs (NN4G) (Micheli 2009). The GNN model is based on a state transition system similar to the RecNN that allows cycles in the state computation, whereas the stability of the recursive encoding process is guaranteed by resorting to a contractive state dynamics (Banach theorem for fixed point), which in turn is obtained by imposing constraints to the loss function (alternating learning and convergence of the recursive dynamical system). In this approach, the context of each vertex is formed through graph diffusion during the iteration to the convergence of state dynamics. Theoretical approximation capability and VC dimension of GNN have been recently studied (Scarselli, Tsoi, and Hagenbuchner 2018).

The NN4G exploits the idea to treat the mutual dependencies among state variables managing them (architecturally) through different layers. without the use of recursive units (also providing an automatic *divide et impera* approach for the architectural design). Instead of iterating at the same layer, each vertex can take the context of the other vertices computed in the previous layers, accessing progressively to the entire graph context. Such idea for a compositional embedding of vertices have been successively exploited in many forms in the area of spatial approaches, or convolutional NN (CNN) for graphs, which all share the process of traversing the graphs by neural units with weight sharing (i.e. for CNN the weights are constrained to the neighbor topology of all graph vertices instead of the 2D matrix), and the construction through many layers moving to deep architectures (Micheli 2009; Zhang et al. 2018; Tran, Navarin, and Sperduti 2018; Atwood and Towsley 2016; Niepert, Ahmed, and Kutzkov 2016; Xu et al. 2018). There are many other approaches in the field, including spectral-based NN approaches (Defferrard, Bresson, and Vandergheynst 2016) and kernel for graphs (Shervashidze et al. 2009; Ya-

Copyright © 2020, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

nardag and Vishwanathan 2015; Vishwanathan et al. 2010; Neumann et al. 2016; Shervashidze et al. 2011).

Summarizing, and following the general trend, learning in structured domains has progressively moved from flat to more structured forms of data (sequences, trees and up to general graphs), while NN models have been extended from shallow to deep architectures, allowing a multi-level abstraction of the input representation. Unfortunately, both aspects imply a high computational cost, highlighting the need to move toward deep *and* efficient approaches for graphs.

For the case of sequences and trees, the Reservoir Computing (RC) paradigm provides an approach for the efficient modeling of recurrent/recursive models based on the use of fixed (randomized) values of the recurrent weights under stability conditions of the dynamical system (Echo State Property - ESP) (Jaeger and Haas 2004; Gallicchio and Micheli 2013). Advantages for a deep construction of RC models in the sequential domain have been analyzed under different points of view, including the richness of internal representations, the occurrence of multiple time-scales (following the architectural hierarchy) and the competitive empirical results, see, e.g., (Gallicchio, Micheli, and Pedrelli 2018). Extension of the RC to graphs (Gallicchio and Micheli 2010) follows the line of GNN, whereas the stability condition is inherited by the contractivity of the reservoir dynamics (ESP). However, the extension of such approaches to multi-layer architectures is still unexplored for general graphs.

Following all these lines, the aim of this work is to provide an approach to graph classification, called Fast and Deep Graph Neural Network (FDGNN), which combines: (i) the capability of stable dynamic systems (in the class of GNN models) for the *graphs embedding*, (ii) the potentiality of a *deep* organization of the GNN architecture (providing hierarchical abstraction of the input structures through the architectural layers), and (iii) the extreme *efficiency* of a randomized neural network, i.e. a model without training of the recurrent units.

The central idea is to exploit the fixed point of the recursive/dynamical system to represent (or *embed*) the input graphs. In particular, once the states have been computed for all graph vertices, iterating the state transition function until convergence, such information is exploited for the graph embedding and then projected to the model output, which is implemented as a standard layer of trained neural units.

The efficiency issue will be addressed also in terms of sparse connections and a relative small number of units compared to the typical setting of RC models or compared to the number of free-parameters of fully-trained approaches. It is worth to note that, since that free-parameters of the embedding part do not undergo a training process, the model also provides a tool to investigate the inherent (i.e. independent from learning) architectural effect of layering in GNN. Hence, such investigation can also provide insights for the set-up of more complex fully-trained models.

The paper also includes a theoretical analysis of the condition for the stability of the neural graph embedding in deep GNN architectures. The experimental analysis includes a significant set of well-known benchmarks in the field of graph classification, hence enabling a thorough comparison with recent approaches at the state-of-the-art.

2 Proposed Method

Preliminaries on graphs. In this paper we deal with graph classification problems. A graph G is represented as a tuple $G = (V_G, E_G)$, where V_G is the set of vertices (or nodes) and $E \subseteq V_G \times V_G$ is the set of edges. The number of vertices in G is denoted by N_G. The connectivity among the vertices in a given graph G is represented by the adjacency matrix $\mathbf{A}_G \in \mathbb{R}^{N_G \times N_G}$, where $\mathbf{A}_G(i,j) = 1$ if there is an edge between vertex i and vertex j, and $\mathbf{A}_G(i,j) = 0$ otherwise. Although the proposed FDGNN approach can be used to process both directed and undirected graphs, in what follows we assume undirected graph structures, i.e., graphs for which the adjacency matrix is a symmetric matrix. The set of vertices adjacent to $v \in V_G$ is the neighborhood of v, i.e., $\mathcal{N}(v) = \{v' \in V_G : (v, v') \in E_G\}$. We use k to indicate the degree of a set of graphs under consideration, i.e. the maximum among the sizes of the neighborhoods of the vertices. Each vertex v is featured by a label, denoted by $\mathbf{l}(v)$, and which we consider to lie in a real vector space. When considering input graphs, we use I to denote the size of vertex labels, i.e., $\mathbf{l}(v) \in \mathbb{R}^I$. In what follows, when the reference to the graph in question is unambiguous, we drop the subscript G to ease the notation.

The neural encoding process. The proposed FDGNN model is based on constructing progressively more abstract neural representation of input graphs by stacking successive layers of GNNs. Crucially, and differently from the original formulation by the proponents of GNN (Scarselli et al. 2009), in FDGNN the parameters of each layers (i.e., the weights pertaining to the hidden units) are left untrained after initialization under stability constraints.

We use L to denote the number of layers in the architecture. Then, for each layer $i = 1, \ldots, L$, the developed neural embedding (or state) for vertex v is indicated by $\mathbf{x}^{(i)}(v)$. This is computed by the state transition function in layer i, which is implemented by neurons featured by hyperbolic tangent non-linearity, as follows:

$$\mathbf{x}^{(i)}(v) = \tanh\big(\mathbf{W}_I^{(i)}\mathbf{u}^{(i)}(v) + \sum_{v' \in \mathcal{N}(v)} \mathbf{W}_H^{(i)}\mathbf{x}^{(i)}(v')\big), \tag{1}$$

where, in relation to the given layer i, $\mathbf{u}^{(i)}(v) \in \mathbb{R}^{U^{(i)}}$ is the input, $\mathbf{W}_I^{(i)} \in \mathbb{R}^{H^{(i)} \times U^{(i)}}$ is the input weight matrix that modulates the influence of the input on the representation, and $\mathbf{W}_H^{(i)} \in \mathbb{R}^{H^{(i)} \times H^{(i)}}$ is the recursive weight matrix that determines the influence of the neighbors representations in the embedding computed for v. Here, we use $U^{(i)}$ and $H^{(i)}$, respectively, to denote the dimension of the input and embedding representations. Moreover, here, and in the rest of the paper, references to bias terms are dropped in the equations for the ease of notation. For all vertices v, the embeddings $\mathbf{x}^{(i)}(v)$ are inizialized to zero values for all i. Following the layered construction, for every vertex v in the input graph, the input information for the first layer is given by the label attached to v, i.e., $\mathbf{u}^{(1)}(v) = \mathbf{l}(v)$ (and $U^{(1)} = I$). For successive layers $i > 1$, the role of input information

for vertex v is played by the embedding developed for v by the previous layer in the stack, i.e., $\mathbf{u}^{(i)}(v) = \mathbf{x}^{(i-1)}(v)$ (and $U^{(i)} = H^{(i-1)}$). From the perspective of the analysis recently introduced in (Xu et al. 2018), FDGNN uses a neighborhood aggregation scheme that corresponds to a sum operator with untrained weights (summation term in eq. 1). Figure 1 shows the deep encoding process focused on a vertex v of the input graph.

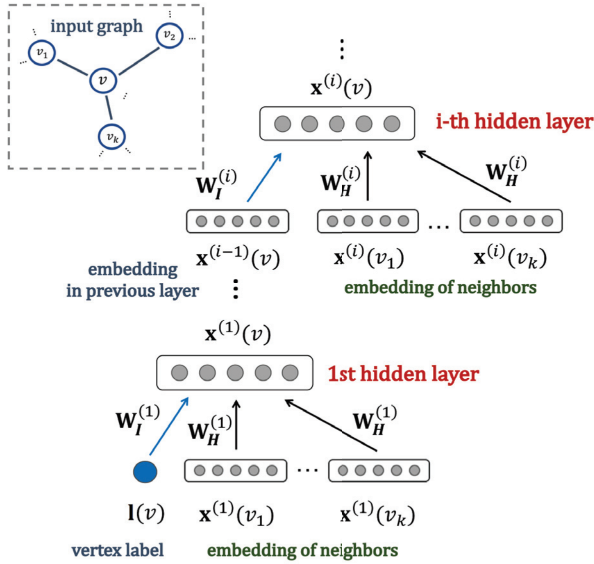

Figure 1: Deep encoding process implemented by FDGNN.

For any given layer $i \in \{1, \ldots, L\}$, the same state transition function in eq. 1 is applied in correspondence of every vertex v of an input graph. The resulting operation can be conveniently and more compactly represented by resorting to a collective notation for the input and the embedding information, i.e., using $\mathbf{U}^{(i)} \in \mathbb{R}^{U^{(i)} \times N}$ and $\mathbf{X}^{(i)} \in \mathbb{R}^{H^{(i)} \times N}$ to column-wise collect respectively the input vectors $\mathbf{u}^{(i)}(v)$ and the embeddings $\mathbf{x}^{(i)}(v)$, for every $v \in V$. In this way, the operation of the i-th hidden layer in the FDGNN architecture can be described by means of a function $F^{(i)}$: $\mathbb{R}^{U^{(i)} \times N} \times \mathbb{R}^{H^{(i)} \times N} \rightarrow \mathbb{R}^{H^{(i)} \times N}$, defined as follows:

$$\mathbf{X}^{(i)} = F^{(i)}(\mathbf{U}^{(i)}, \mathbf{X}^{(i)}) = \tanh(\mathbf{W}_I^{(i)}\mathbf{U}^{(i)} + \mathbf{W}_H^{(i)}\mathbf{X}^{(i)}\mathbf{A}). \quad (2)$$

In case of mutual dependencies among the vertices in the input graph (e.g., in the presence of cyclic substructures or of undirected connections) the application of eq. 2 (or, equivalently of eq. 1 in a vertex-wise fashion), might not admit a unique solution, i.e, a valid graph embedding representation. It is then useful to study eq. 2 as the iterated map that rules a discrete-time non-linear dynamical system, where $\mathbf{X}^{(i)}$ plays the role of state information, and $\mathbf{U}^{(i)}$ acts as the driving input. To ensure the uniqueness of the developed neural representations, it is important that the system described by eq. 2 is asymptotically stable, i.e., it converges to a unique fixed point upon repeated iteration. Notice that such fixed point depends on both the input labels (i.e., $\mathbf{U}^{(i)}$) and the topology (i.e., the adjacency matrix $\mathbf{A}$) of each input graph. The system dynamics are parametrized by the weight values in $\mathbf{W}_I^{(i)}$ and $\mathbf{W}_H^{(i)}$, which play a fundamental role in characterizing the resulting dynamical behavior. In the standard GNN formulation, the neural network alternates phases of state relaxation and learning of $\mathbf{W}_I^{(i)}$ and $\mathbf{W}_H^{(i)}$ subject to contractive constraint employed through a penalizer term in the loss function for gradient descent (Scarselli et al. 2009). In this paper, instead of employing a costly (and possibly long) constrained training process, we study stability conditions for the parameters in eq. 2, and initialize them accordingly. After initialization, the weights in $\mathbf{W}_I^{(i)}$ and $\mathbf{W}_H^{(i)}$ are left untrained, resulting in a graph encoding process that is extremely faster than that of classical GNNs, and is able to develop rich neural embeddings in a hierarchical fashion. The stability conditions for FDGNN initialization are described in depth in Section 2.1. As a further characterizing aspect of our approach, we make use of a *sparse* design for both matrices $\mathbf{W}_H^{(i)}$ and $\mathbf{W}_I^{(i)}$. In particular, for every neuron in each hidden layer, we have only a constant number C of connections from the previous layer (or from the input, for the first hidden layer), and of recurrent connections from the same layer. Note that the topology of the hidden layer architecture, i.e. the pattern of connectivity among the neurons, does not need to be related to the topology of the input graph. In particular, each neuron takes inputs from all the neighbour vertices embeddings (up to the degree k) from a number of neurons depending on C (that controls the sparsity of connectivity in the NN layer). The topology of the input graph is hence preserved by the embedding process.

Based on the above described formulation, for any given input graph, the FDGNN encoding process proceeds as follows. The state transition that rules the first hidden layer, i.e., $F^{(1)}$, is iterated until convergence to its fixed point, i.e., $\mathbf{X}^{(1)}$, driven by the vertices labels information as external input. Then, the same operation is repeated for the second layer, whose dynamics are ruled by $F^{(2)}$ and driven by $\mathbf{X}^{(1)}$, which now plays the role of input. This process goes on until the last layer L is reached, and its state transition function $F^{(L)}$ converges to its fixed point $\mathbf{X}^{(L)}$. At each layer, the convergence to the fixed point is stopped whenever the distance between the states in successive iterations is below a threshold ϵ, or a maximum number of iterations ν is reached.

Output computation. For a given graph, the output of FDGNN is computed by means of a simple readout layer that combines the neural representations developed by the last hidden layer of the architecture, as follows:

$$\mathbf{y} = \mathbf{W}_Y \ \tanh\big(\mathbf{W}_\varphi \sum_{v \in V} \mathbf{x}^{(L)}(v)\big), \quad (3)$$

where the argument of the hyperbolic tangent non-linearity expresses a combination of the embeddings computed in the last hidden layer for all the vertices in the graph, modulated by a projection matrix $\mathbf{W}_\varphi \in \mathbb{R}^{P \times H^{(i)}}$. Finally, $\mathbf{W}_Y \in \mathbb{R}^{Y \times P}$ is the output weight matrix, where $\mathbb{R}^Y$ is the output space. In this work, elements in $\mathbf{W}_\varphi$ are randomly initialized from a uniform distribution and then are

re-scaled and controlled to have unitary L_2-norm. The elements in $\mathbf{W}_Y$ are the only free parameters of the model that undergo a training process, and are adjusted based on the training set. In the vein of RC approaches, training of the output weights is performed (non-iteratively) in closed form, exploiting the resulting convexity of the learning problem. This is performed by using Tikhonov regularization as described in (Lukoševičius and Jaeger 2009). The readout operation implemented by eq. 3 can be seen as the application of a sum-pooling (or aggregation) operation, followed by a randomized non-linear basis expansion, and finally by a layer of trained linear neurons.

2.1 Stability Conditions for Graph Embedding

We study dynamical stability of the system implemented by each hidden layer of a GNN, as reported in eq. 2. Here we focus our analysis on a generic layer i of the architecture, and drop the (i) superscript in the mathematical objects for the ease of notation. Accordingly, the dynamics under consideration are governed by the non-linear map F that, given the input for the layer $\mathbf{U}$, and an initial state $\mathbf{X}_0$ (set to zero values), is iterated until convergence to the fixed-point solution, i.e. to the attractor of the dynamics.

Here we use F_t to denote the t-th iterate of F, and $\mathbf{X}_t$ to indicate the state after t iterations, i.e., $\mathbf{X}_t = F(\mathbf{U}, \mathbf{X}_{t-1}) = F(\mathbf{U}, F(\mathbf{U}, \mathbf{X}_{t-2})) = \ldots = F(\mathbf{U}, F(\mathbf{U}, F(\ldots(F(\mathbf{U}, \mathbf{X}_0))\ldots)))$. In the following, we assume that input and state spaces are compact sets, moreover we use $\|\cdot\|$ to indicate L_2-norm, and $\rho(\cdot)$ to indicate the spectral radius[1]. In order to develop a valid neural representation of input graphs, we require the system ruled by F to be globally asymptotically stable, according to the following definition of *Graph Embedding Stability* (GES).

Definition 1 (Graph Embedding Stability (GES))**.** *For every input $\mathbf{U}$ to the current layer, and for every $\mathbf{X}_0, \mathbf{Z}_0$ initial states for the neural embeddings in the current layer, it results that:*

$$\|F_t(\mathbf{U}, \mathbf{X}_0) - F_t(\mathbf{U}, \mathbf{Z}_0)\| \to 0 \quad as \quad t \to \infty. \qquad (4)$$

GES in Definition 1 essentially says that irrespective of the initial conditions, the developed neural encoding for the same input graph should be unique. Perturbations should vanish as the convergence process proceeds, and the resulting graph encoding is robust. The property expressed by the GES can be also seen as a generalization of the ESP commonly imposed in the context of RC (Jaeger and Haas 2004). In the same vein as studies in RC, here we provide two conditions for the GES of the hidden layers in deep GNN, one sufficient, expressed by Theorem 1, and one necessary, expressed by Theorem 2.

Theorem 1 (Sufficient condition for GES)**.** *For every input $\mathbf{U}$ to the current layer, if $\|\mathbf{W}_H\|\, k < 1$ then F has dynamics that satisfy the GES.*

Theorem 2 (Necessary condition for GES)**.** *Assume that a k-regular graph with null vertices labels is an admissible input for the system. If F has dynamics that satisfy the GES then $\rho(\mathbf{W}_H)\, k < 1$.*

[1] The largest abs value of the eigenvalues of its matrix argument.

The proofs of both Theorems 1 and 2 are reported in the Supplementary Material (Gallicchio and Micheli 2019).

Theorems 1 and 2 provide a grounded way for stable initialization of hidden layers' in FDGNN. In particular, the sufficient condition corresponds to a contractive setting of the neural dynamics for every possible input, and hence could be too restrictive in practical applications. Accordingly, we adopt the condition implied by Theorem 2 to initialize the weights in the hidden layers of the FDGNN, generalizing a common practice in the RC field (Jaeger and Haas 2004). As such, for every layer $i = 1, \ldots, L$, weight values in $\mathbf{W}_H^{(i)}$ are first randomly chosen from a uniform distribution in $[-1, 1]$ and then re-scaled to have an effective spectral radius $\rho^{(i)} = \rho(\mathbf{W}_H^{(i)})\, k$, such that $\rho^{(i)} < 1$. As pertains to the input matrices $\mathbf{W}_I^{(i)}$, their values are randomly sampled from a uniform distribution in $[-\omega^{(i)}, \omega^{(i)}]$. The value of $\omega^{(i)}$ acts as input scaling for $i = 1$, and as inter-layer scaling for $i > 1$. We treat $\rho^{(i)}$ and $\omega^{(i)}$ as hyper-parameters.

2.2 Computational Cost

For any given input graph, the cost of the graph embedding process at layer i is that of the iterated application of eq. 2. Assuming H hidden neurons, and exploiting the sparse connectivity in the hidden layer matrices, each iteration requires $\mathcal{O}((C + k)\, H\, N)$. As such, the entire process of graph embedding for the whole architecture has a cost that is given by $\mathcal{O}(L\, \nu\, (C + k)\, H\, N)$, which scales linearly with the number of neurons, with the number of layers, with the degree, and with the number of vertices (i.e., the size of the input graph). Strikingly, the cost of the encoding is the same for both training and test, as the internal weights do not undergo a training process and hence no additional cost for training is required. This leads to a clear advantage in comparison to fully trained NN models for graphs. Besides, the efficiency of the proposed approach is comparable to the Weisfeiler-Lehman graph kernel (Shervashidze et al. 2011), one of the most known and fastest kernels for graphs, whose cost scales linearly with the number of vertices and edges.

The efficiency of FDGNN emerges also in the process of output computation, which is performed by a simple readout tool. This can be trained efficiently using direct methods, and generally is less expensive than training alternative readout implementations requiring gradient descent learning possibly through multiple layers (e.g., as in NNs for graphs), or support vector machines (e.g., as in kernel for graphs).

3 Experiments

We experimentally assess the proposed FDGNN model on several benchmark datasets for graph classification, in comparison to state-of-the-art approaches from literature.

Datasets. We consider 9 graph classification benchmarks from the areas of cheminformatics and social network analysis. All the used datasets are publicly available online (Kersting et al. 2016). In the case of cheminformatics datasets, input graphs are used to represent chemical compounds, where vertices stand for atoms and are labeled by the atom type (represented by one-hot encoding), while edges be-

tween vertices represent bonds between the corresponding atoms. In particular, MUTAG (Debnath et al. 1991) is a collection of nitroaromatic compounds and the goal is to predict their mutagenicity on Salmonella typhimurium, PTC (Helma et al. 2001) is a set of chemical compounds that are classified as carcinogenic or non-carcinogenic for male rats, COX2 (Sutherland, O'brien, and Weaver 2003) contains a set of cyclooxygenase-2 inhibitor molecules that are labeled as active or inactive, PROTEINS (Borgwardt et al. 2005) is a dataset of proteins that are classified as enzymes or non-enzymes, NCI1 (Wale, Watson, and Karypis 2008) contains anti-cancer screens for cell lung cancer. As regards the social network domain, we use IMDB-BINARY (IMDB-b), IMDB-MULTI (IMDB-m), REDDIT (binary) and COLLAB, proposed in (Yanardag and Vishwanathan 2015). IMDB-b and IMDB-m are movie collaboration datasets containing actor/actress ego-networks where the target class represent the movie genre, with 2 possible classes for IMDB-b (i.e., action or romance), and 3 classes for IMDB-m (i.e., comedy, romance, and sci-fi). REDDIT is a collection of graphs corresponding to online discussion threads, classified in 2 classes (i.e., question/answer or discussion community). Finally, COLLAB is a set of scientific collaborations, containing ego-networks of researchers classified based on the area of research (i.e., high energy, condensed matter or astro physics). Differently from the case of cheminformatics benchmarks, for the social network datasets no label is attached to the vertices in the graph. In this cases, we use a fixed (uni-dimensional) input label equal to 1 for all the vertices. For binary classification tasks, a target value in $\{-1, 1\}$ is considered. For multi-class classification tasks, the target label for each graph is a vector representing a binary $-1/+1$ one-hot encoding of the corresponding class. A summary of datasets information is in the Supplementary Material (Gallicchio and Micheli 2019).

Experimental settings. We adopted a simple general setting, where all the hidden layers of the architecture in the graph embedding component shared the same values of the hyper-parameters, i.e., for $i \geq 1$ we set: number of neurons $H^{(i)} = H$ and effective spectral radius $\rho^{(i)} = \rho$; for $i > 1$ we set the inter-layer scaling $\omega^{(i)} = \omega$. In particular, we set the number of neurons in each hidden layer to $H = 50$ for all datasets except for NCI1 and COLLAB, for which we used $H = 500$. We implemented very sparse weight matrices with $C = 1$, i.e. every hidden layer neuron receives only 1 incoming connection from the previous layer (i.e., the input layer for neurons in the first hidden layer), and 1 incoming recurrent connection from a neuron in the same hidden layer. Values of ρ, $\omega^{(1)}$ and ω were explored in the same range $(0, 1)$. The above hyper-parameters of the graph embedding component were explored by random search, i.e., by randomly generating a number of 100 configurations within the reported ranges. For every configuration, the value of the Tikhonov regularizer for training the readout was searched in a log-scale grid in the range $10^{-8} - 10^{3}$. To account for randomization, for every configuration we instantiated 20 networks guesses, averaging the error on such guesses. For the graph embedding convergence process in each layer we used a threshold of $\epsilon = 10^{-3}$, and maximum number of iterations $\nu = 50$. The projection dimension for the readout was set to twice the number of neurons in the last hidden layer, i.e., $P = 2\ H$. Accordingly, the total number of free parameters, i.e., the number of trainable weights for the learner, is as small as 1001 for NCI1 and COLLAB, and 101 for all the other datasets (note that there is an additional weight for the output bias). For all the datasets, we used the average value of k in the dataset for network initialization.

For binary classification tasks, the output class for each graph was computed using the readout equation in eq. 3, followed by the sign function for $-1/+1$ discretization. In multi-class classification tasks, to each graph was assigned an output class in correspondence of the readout unit with the highest activation. The performance on the graph classification tasks was assessed in terms of accuracy, and it was evaluated through a process of stratified 10-fold cross validation. For each fold, the FDGNN hyper-parameter configuration was chosen by model selection, by means of a nested level of stratified 10-fold cross validation applied on the corresponding training samples.

3.1 Results

The results achieved by FDGNN are reported in Table 1, where we show the test accuracy averaged over the outer 10 folds of the cross-validation (std are reported on the folds). We also give the performance achieved by FDGNN settings constrained to a single hidden layer ($L = 1$). For comparison, in the same table we report the accuracy obtained by literature ML models for graphs. These includes a variety of NN for graphs, i.e., GNN (Scarselli et al. 2009), Relational Neural Networks (RelNN) (Blockeel and Bruynooghe 2003), Deep Graph Convolutional Neural Network (DGCNN) (Zhang et al. 2018), Parametric Graph Convolution DGCNN (PGC-DGCNN) (Tran, Navarin, and Sperduti 2018), Diffusion-Convolutional Neural Networks (DCNN) (Atwood and Towsley 2016), PATCHY-SAN (PSCN) (Niepert, Ahmed, and Kutzkov 2016). We also consider a number of relevant models from state-of-the-art graph kernels: Graphlet Kernel (GK) (Shervashidze et al. 2009), Deep GK (DGK) (Yanardag and Vishwanathan 2015), Random-walk Kernel (RW) (Vishwanathan et al. 2010), Propagation Kernel (PK) (Neumann et al. 2016), and Weisfeiler-Lehman Kernel (WL) (Shervashidze et al. 2011). Moreover, we consider two further recently introduced models: a recent hybrid CNN-kernel approach named Kernel Graph CNN (KGCNN) (Nikolentzos et al. 2018), and Contextual Graph Markov Model (CGMM) (Bacciu, Errica, and Micheli 2018), which merges generative models and NN for graphs. The performance in Table 1 for the above mentioned approaches is quoted from the indicated reference (where the experimental setting for model selection was as rigorous as ours), with the exception of PK and WL on COX2, which are quoted from (Neumann et al. 2016). When more than one configuration is reported in the literature reference, we quote the result of the one with the highest accuracy.

Results in Table 1 indicate that FDGNN outperforms the best literature results on 7 out of 9 benchmarks, achieving state-of-the-art performance, and showing in many cases a

Table 1: Test accuracy of FDGNN, compared to state-of-the-art results from literature. Performance of single hidden layer versions of FDGNN ($L = 1$) are reported as well. Results are averaged (and std are computed) on the outer 10 folds. Best results are highlighted in bold for every dataset.

	MUTAG	PTC	COX2	PROTEINS	NCI1
FDGNN	$\mathbf{88.51}_{\pm 3.77}$	$\mathbf{63.43}_{\pm 5.35}$	$\mathbf{83.39}_{\pm 2.88}$	$\mathbf{76.77}_{\pm 2.86}$	$77.81_{\pm 1.62}$
FDGNN$_{(L=1)}$	$87.38_{\pm 6.55}$	$\mathbf{63.43}_{\pm 5.35}$	$82.41_{\pm 2.67}$	$\mathbf{76.77}_{\pm 2.86}$	$77.11_{\pm 1.52}$
GNN (Uwents et al. 2011)	$80.49_{\pm 0.81}$	-	-	-	-
RelNN (Uwents et al. 2011)	$87.77_{\pm 2.48}$	-	-	-	-
DGCNN (Zhang et al. 2018)	$85.83_{\pm 1.66}$	$58.59_{\pm 2.47}$	-	$75.54_{\pm 0.94}$	$74.44_{\pm 0.47}$
PGC-DGCNN (Tran, Navarin, and Sperduti 2018)	$87.22_{\pm 1.43}$	$61.06_{\pm 1.83}$	-	$76.45_{\pm 1.02}$	$76.13_{\pm 0.73}$
DCNN (Tran, Navarin, and Sperduti 2018)	-	-	-	$61.29_{\pm 1.60}$	$56.61_{\pm 1.04}$
PSCN (Tran, Navarin, and Sperduti 2018)	-	-	-	$75.00_{\pm 2.51}$	$76.34_{\pm 1.68}$
GK (Zhang et al. 2018)	$81.39_{\pm 1.74}$	$55.65_{\pm 0.46}$	-	$71.39_{\pm 0.31}$	$62.49_{\pm 0.27}$
DGK (Yanardag and Vishwanathan 2015)	$82.66_{\pm 1.45}$	$57.32_{\pm 1.13}$	-	$71.68_{\pm 0.50}$	$62.48_{\pm 0.25}$
RW (Zhang et al. 2018)	$79.17_{\pm 2.07}$	$55.91_{\pm 0.32}$	-	$59.57_{\pm 0.09}$	-
PK (Zhang et al. 2018)	$76.00_{\pm 2.69}$	$59.50_{\pm 2.44}$	$81.00_{\pm 0.20}$	$73.68_{\pm 0.68}$	$82.54_{\pm 0.47}$
WL (Zhang et al. 2018)	$84.11_{\pm 1.91}$	$57.97_{\pm 2.49}$	$83.20_{\pm 0.20}$	$74.68_{\pm 0.49}$	$\mathbf{84.46}_{\pm 0.45}$
KCNN (Nikolentzos et al. 2018)	-	$62.94_{\pm 1.69}$	-	$75.76_{\pm 0.28}$	$77.21_{\pm 0.22}$
CGMM (Bacciu, Errica, and Micheli 2018)	85.30	-	-	-	-

	IMDB-b	IMDB-m	REDDIT	COLLAB
FDGNN	$\mathbf{72.36}_{\pm 3.63}$	$\mathbf{50.03}_{\pm 1.25}$	$\mathbf{89.48}_{\pm 1.00}$	$74.44_{\pm 2.02}$
FDGNN$_{(L=1)}$	$71.79_{\pm 3.37}$	$49.34_{\pm 1.70}$	$87.74_{\pm 1.61}$	$73.82_{\pm 2.32}$
DGCNN (Zhang et al. 2018)	$70.03_{\pm 0.86}$	$47.83_{\pm 0.85}$	-	$73.76_{\pm 0.49}$
PGC-DGCNN (Tran, Navarin, and Sperduti 2018)	$71.62_{\pm 1.22}$	$47.25_{\pm 1.44}$	-	$\mathbf{75.00}_{\pm 0.58}$
PSCN (Tran, Navarin, and Sperduti 2018)	$71.00_{\pm 2.29}$	$45.23_{\pm 2.84}$	-	$72.60_{\pm 2.15}$
GK (Yanardag and Vishwanathan 2015)	$65.87_{\pm 0.98}$	$43.89_{\pm 0.38}$	$77.34_{\pm 0.18}$	$72.84_{\pm 0.56}$
DGK (Yanardag and Vishwanathan 2015)	$66.96_{\pm 0.56}$	$44.55_{\pm 0.52}$	$78.04_{\pm 0.39}$	$73.09_{\pm 0.25}$
KCNN (Nikolentzos et al. 2018)	$71.45_{\pm 0.15}$	$47.46_{\pm 0.21}$	$81.85_{\pm 0.12}$	$74.93_{\pm 0.14}$

clear improvement with respect to ML models in the area of NNs, kernel for graphs, and hybrid variants. Moreover, even in the cases where FDGNN does not achieve the top performance, its accuracy results to be the highest within the class of NN for graphs (for NCI1), or it is very close to the highest one (for COLLAB).

Results reported in Table 1 are surprising, especially considering that they are obtained by a model in which the graph embedding process is not subject to training. Relevantly, the experimental results illustrated here highlight the ability of layered GNN architectures to construct neural embeddings for graph data that are rich in an *intrinsic* way, i.e., even in the absence (or before) training of the recurrent connections. Complementarily, our results point out the important role played by *stability* of the graph encoding process: as long as the layers of a deep GNN are able to implement such a process in a stable way, the neural representations developed in the hidden layers are *per se* useful to effectively solve real-world problems in the area of graph classification.

The impact of randomization in the FDGNN weight matrices initialization can be quantified by the std on the instances (i.e., the guesses), reported in the following: MUTAG (2.39), PTC (4.56), COX2 (2.50), PROTEINS (1.72), NCI1 (1.59), IMDB-b (1.39), IMDB-m (1.35), REDDIT (2.33), COLLAB (1.03). These results indicate that the variance on the instances is in line with - and is generally smaller than - the variance on the folds (given in Table 1).

Table 1 also indicates the advantage of depth in the architectural setup of FDGNN. While the accuracy achieved by single hidden layer settings (i.e., for $L = 1$) is already comparable to literature results, deeper settings of FDGNN consistently obtain a better performance (with the only exceptions of PTC and PROTEINS, where 1 hidden layer configurations were selected by model selection). As a further experimental reference, Table 2 shows the depth of the selected FDGNN configuration for each dataset, averaged on the 10 folds. In most cases a number of layers of ≈ 3 (and up to 5) is selected, testifying the effectiveness of the multi-layered construction of the FDGNN architecture.

The use of untrained recurrent connections makes the proposed FDGNN strikingly efficient in applications. As evidence of this, in Table 3 we report the running times for the training and test of the selected FDGNN configurations, averaged on the 10 folds. The reported times are referred to

Table 2: Depth of FDGNN configurations selected by the nested cross validation process. Results are averaged (and std are computed) on the outer 10 folds.

MUTAG	PTC	COX2	PROTEINS	NCI1
$3.2_{\pm 1.0}$	$1.0_{\pm 0.0}$	$2.7_{\pm 0.8}$	$1.0_{\pm 0.0}$	$3.8_{\pm 0.6}$
IMDB-b	**IMDB-m**	**REDDIT**	**COLLAB**	
$3.2_{\pm 1.2}$	$4.4_{\pm 0.8}$	$3.0_{\pm 0.0}$	$4.5_{\pm 0.8}$	

Table 3: Running times required by FDGNN (in single core mode, without GPU acceleration). Results are averaged (and std are computed) on the outer 10 folds. Times are reported in seconds ($''$) or minutes ($'$).

Task	Training	Test
MUTAG	$0.56''_{\pm 0.33}$	$0.06''_{\pm 0.04}$
PTC	$0.16''_{\pm 0.03}$	$0.02''_{\pm 0.00}$
COX2	$1.36''_{\pm 0.42}$	$0.15''_{\pm 0.05}$
PROTEINS	$2.16''_{\pm 0.47}$	$0.24''_{\pm 0.04}$
NCI1	$2.00'_{\pm 0.45}$	$13.36''_{\pm 3.02}$
IMDB-b	$7.46''_{\pm 3.14}$	$0.83''_{\pm 0.35}$
IMDB-m	$8.68''_{\pm 1.73}$	$0.98''_{\pm 0.22}$
REDDIT	$2.47'_{\pm 0.01}$	$16.49''_{\pm 0.28}$
COLLAB	$22.86'_{\pm 4.70}$	$2.54'_{\pm 0.52}$

a MATLAB implementation of FDGNN, running on a system with Intel Xeon Processor E5-263L v3 with 168 GB of RAM. Note that, although the FDGNN model is amenable for parallelization, to provide an unbiased estimation of the required computation time, the reported running times were obtained using the system in single-core mode, and without any GPU acceleration. Our code is made available online[2]. As it can be noticed, training and test times of FDGNN (even without parallelization) are generally very low, resulting from the combination of a number of factors: the sparse design of the hidden layers' matrices, the small number of neurons in each hidden layer, and the fact the internal weights are left untrained. Moreover, also the iterative embedding stabilization process taking place in each hidden layer is in practice not computationally intensive. Overall the approach results very fast, requiring a very small number of trainable weights (up to 1001 in our experiments), especially in comparison to literature models that entail much more complex architectural settings with multiple layers of fully end-to-end trained neurons, and possibly hundreds of thousands of trainable weights.

Table 4: Comparison of training times required on MUTAG by FDGNN, GNN, GIN and WL. Results are averaged (and std are computed) on the outer 10 folds.

FDGNN	GNN	GIN	WL
$\mathbf{0.56}''_{\pm 0.33}$	$202.28''_{\pm 166.87}$	$499.24''_{\pm 2.25}$	$1.16''_{\pm 0.03}$

[2]https://sites.google.com/site/cgallicch/FDGNN/code

In order to give a more concrete idea on the efficiency of the proposed approach, in Table 4 we compare the training times required by FDGNN on MUTAG with those required by GNN, GIN (Xu et al. 2018), and WL, as representatives respectively of dynamical NNs, recent convolutional NNs, and kernels models for graphs. In all cases, we used the code made available by the proponents, using the hyper-parameter settings given by the authors, and the same machine used for experiments on FDGNNs. Results in Table 4 indicate that FGDNN requires the smallest training times, and results to be more than 300 times faster than GNN, almost 900 times faster than GIN, and comparable to WL (but still ≈ 2 times faster). Relevantly, the FDGNN training speedup pairs the accuracy improvement already shown with respect to GNN and WL in Table 1, where accuracy of GIN from literature is not reported because obtained as validation accuracy of a 10-fold cross validation (while we are uniformly considering a more rigorous nested cross validation with external test set). For completeness, the performance of GIN reported in (Xu et al. 2018) ranges from 0.835 to 0.9 (depending on the variant). In similar experimental conditions, FDGNN achieves a test accuracy of 0.948.

4 Conclusions

We have introduced FDGNN, a novel NN model for fast learning in graph domains. The proposed approach showed that it is possible to combine the advantages of a deep architectural construction of GNN (in its ability to effectively process structured data in the form of general graphs), with the extreme efficiency of randomized NN, and in particular RC, methodologies. The randomized implementation allows us to implement untrained - but *stable* - graph embedding layers, while through the deep architecture the model is able to build a progressively more effective representation of the input graphs. Despite the simplicity of the setup and the fast computation allowed by the model, the empirical accuracy of FDGNN results to be very competitive with a large number of state-of-the-art CNN models and kernel methods for graphs. Further possible analysis on the expressive power of FDGNN can be related to the study of Markovian-based organization of graph embedding spaces.

References

Atwood, J., and Towsley, D. 2016. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems*, 1993–2001.

Bacciu, D.; Errica, F.; and Micheli, A. 2018. Contextual graph markov model: A deep and generative approach to graph processing. In *Proceedings of ICML 2018*.

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

Borgwardt, K. M.; Ong, C. S.; Schönauer, S.; Vishwanathan, S.; Smola, A. J.; and Kriegel, H.-P. 2005. Protein function prediction via graph kernels. *Bioinformatics* 21(suppl_1):i47–i56.

Debnath, A. K.; Lopez de Compadre, R. L.; Debnath, G.; Shusterman, A. J.; and Hansch, C. 1991. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry* 34(2):786–797.

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, 3844–3852.

Frasconi, P.; Gori, M.; and Sperduti, A. 1998. A general framework for adaptive processing of data structures. *IEEE Transactions on Neural Networks* 9(5):768–786.

Gallicchio, C., and Micheli, A. 2010. Graph echo state networks. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, 1–8. IEEE.

Gallicchio, C., and Micheli, A. 2013. Tree echo state networks. *Neurocomputing* 101:319–337.

Gallicchio, C., and Micheli, A. 2019. Fast and Deep Graph Neural Networks. *arXiv preprint arXiv:1911.08941*.

Gallicchio, C.; Micheli, A.; and Pedrelli, L. 2018. Design of deep echo state networks. *Neural Networks* 108:33–47.

Helma, C.; King, R. D.; Kramer, S.; and Srinivasan, A. 2001. The predictive toxicology challenge 2000–2001. *Bioinformatics* 17(1):107–108.

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

Kersting, K.; Kriege, N. M.; Morris, C.; Mutzel, P.; and Neumann, M. 2016. Benchmark data sets for graph kernels.

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

Micheli, A.; Sona, D.; and Sperduti, A. 2004. Contextual processing of structured data by recursive cascade correlation. *IEEE Transactions on Neural Networks* 15(6):1396–1410.

Micheli, A. 2009. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks* 20(3):498–511.

Neumann, M.; Garnett, R.; Bauckhage, C.; and Kersting, K. 2016. Propagation kernels: efficient graph kernels from propagated information. *Machine Learning* 102(2):209–245.

Niepert, M.; Ahmed, M.; and Kutzkov, K. 2016. Learning convolutional neural networks for graphs. In *International conference on machine learning*, 2014–2023.

Nikolentzos, G.; Meladianos, P.; Tixier, A. J.-P.; Skianis, K.; and Vazirgiannis, M. 2018. Kernel graph convolutional neural networks. In *International Conference on Artificial Neural Networks*, 22–32. Springer.

Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2009. The graph neural network model. *IEEE Transactions on Neural Networks* 20(1):61–80.

Scarselli, F.; Tsoi, A. C.; and Hagenbuchner, M. 2018. The vapnik–chervonenkis dimension of graph and recursive neural networks. *Neural Networks* 108:248–259.

Shervashidze, N.; Vishwanathan, S.; Petri, T.; Mehlhorn, K.; and Borgwardt, K. 2009. Efficient graphlet kernels for large graph comparison. In *Artificial Intelligence and Statistics*, 488–495.

Shervashidze, N.; Schweitzer, P.; Leeuwen, E. J. v.; Mehlhorn, K.; and Borgwardt, K. M. 2011. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research* 12(Sep):2539–2561.

Socher, R.; Lin, C. C.; Manning, C.; and Ng, A. Y. 2011. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, 129–136.

Socher, R.; Perelygin, A.; Wu, J.; Chuang, J.; Manning, C. D.; Ng, A.; and Potts, C. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, 1631–1642.

Sperduti, A., and Starita, A. 1997. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks* 8(3):714–735.

Sutherland, J. J.; O'brien, L. A.; and Weaver, D. F. 2003. Spline-fitting with a genetic algorithm: A method for developing classification structure- activity relationships. *Journal of chemical information and computer sciences* 43(6):1906–1915.

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

Uwents, W.; Monfardini, G.; Blockeel, H.; Gori, M.; and Scarselli, F. 2011. Neural networks for relational learning: an experimental comparison. *Machine Learning* 82(3):315–349.

Vishwanathan, S. V. N.; Schraudolph, N. N.; Kondor, R.; and Borgwardt, K. M. 2010. Graph kernels. *Journal of Machine Learning Research* 11(Apr):1201–1242.

Wale, N.; Watson, I. A.; and Karypis, G. 2008. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems* 14(3):347–375.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2018. How powerful are graph neural networks? In *Proceedings of ICLR 2019*.

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

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