Machine Learning (2022) 111:1205–1237
https://doi.org/10.1007/s10994-021-06098-0

Polynomial-based graph convolutional neural networks for graph classification

Luca Pasa[1,2] · **Nicolò Navarin**[1,2] · **Alessandro Sperduti**[1,2]

Received: 31 January 2021 / Revised: 22 July 2021 / Accepted: 8 October 2021 /
Published online: 9 November 2021

Abstract
Graph convolutional neural networks exploit convolution operators, based on some neighborhood aggregating scheme, to compute representations of graphs. The most common convolution operators only exploit local topological information. To consider wider topological receptive fields, the mainstream approach is to non-linearly stack multiple graph convolutional (GC) layers. In this way, however, interactions among GC parameters at different levels pose a bias on the flow of topological information. In this paper, we propose a different strategy, considering a single graph convolution layer that independently exploits neighbouring nodes at different topological distances, generating decoupled representations for each of them. These representations are then processed by subsequent readout layers. We implement this strategy introducing the polynomial graph convolution (PGC) layer, that we prove being more expressive than the most common convolution operators and their linear stacking. Our contribution is not limited to the definition of a convolution operator with a larger receptive field, but we prove both theoretically and experimentally that the common way multiple non-linear graph convolutions are stacked limits the neural network expressiveness. Specifically, we show that a graph neural network architecture with a single PGC layer achieves state of the art performance on many commonly adopted graph classification benchmarks.

Keywords Graph convolutional networks · Graph neural network · Deep learning · Structured data · Machine learning on graphs

Editors: Annalisa Appice, Sergio Escalera, Jose A. Gamez, Heike Trautmann.

✉ Luca Pasa
lpasa@math.unipd.it

Nicolò Navarin
nicolo.navarin@unipd.it

Alessandro Sperduti
alessandro.sperduti@unipd.it

[1] Department of Mathematics, University of Padua, Padua, Italy

[2] Human Inspired Technology Research Centre, University of Padua, Padua, Italy

1 Introduction

In the last few years, the definition of machine learning methods, particularly neural networks, for graph-structured inputs has been gaining increasing attention in literature (Defferrard et al. 2016; Errica et al. 2020). In particular, graph convolutional networks (GCNs), based on the definition of a convolution operator in the graph domain, are relatively fast to compute and have shown good predictive performance. Graph convolutions (GC) are generally based on a neighborhood aggregation scheme (Gilmer et al. 2017) considering, for each node, only its direct neighbors. Stacking multiple GC layers, the size of the receptive field of deeper filters increases (resembling standard convolutional networks). However, stacking too many GC layers may be detrimental on the network ability to represent meaningful topological information (Li et al. 2018) due to a too high Laplacian smoothing. Moreover, in this way interactions among GC parameters at different layers pose a bias on the flow of topological information, as we will discuss in this paper. For these reasons, several convolution operators have been defined in literature, differing from one another in the considered aggregation scheme. We argue that the performance of GC networks could benefit by increasing the size of the receptive fields, but since with existing GC architectures this effect can only be obtained by stacking more GC layers, the increased difficulty in training and the limitation of expressiveness given by the stacking of many local layers ends up hurting their predictive capabilities.

Consequently, the performances of existing GCNs are strongly dependent on the specific architecture. Therefore, existing graph neural network performances are limited by (i) the necessity to select a specific convolution operator, and (ii) the limitation of expressiveness caused by large receptive fields being possible only stacking many local layers.

In this paper, we tackle both issues following a different strategy. We propose the polynomial graph convolution (PGC) layer that independently considers neighbouring nodes at different topological distances (i.e. arbitrarily large receptive fields). We show that the PGC layer is more general than most convolution operators in literature. As for the second issue, a single PGC layer, directly considering larger receptive fields, can represent a richer set of functions compared to the *linear stacking* of two or more graph convolution layers, i.e. it is more expressive. Moreover, the linear PGC design allows to consider large receptive fields without incurring in typical issues related to training deep networks. We developed the polynomial graph convolutional network (PGCN), an architecture that exploits the PGC layer to perform graph classification tasks. We empirically evaluate the proposed PGCN on eight commonly adopted graph classification benchmarks. We compare the proposed method to several state-of-the-art GCNs, consistently achieving higher or comparable predictive performances. Differently from other works in literature, the contribution of this paper is to show that the common approach of stacking multiple GC layers may not provide an optimal exploitation of topological information because of the strong coupling of the depth of the network with the size of the topological receptive fields. In our proposal, the depth of the PGCN is decoupled from the receptive field size, allowing to build deep GNNs avoiding the oversmoothing problem.

2 Notation

We use italic letters to refer to variables, bold lowercase to refer to vectors, and bold uppercase letters to refer to matrices. The elements of a matrix $\mathbf{A}$ are referred to as a_{ij} (and similarly for vectors). We use uppercase letters to refer to sets or tuples. Let $G = (V, E, \mathbf{X})$ be a graph, where $V = \{v_0, \ldots, v_{n-1}\}$ denotes the set of vertices (or nodes) of the graph,

$E \subseteq V \times V$ is the set of edges, and $\mathbf{X} \in \mathbb{R}^{n \times s}$ is a multivariate signal on the graph nodes with the i-th row representing the attributes of v_i. We define $\mathbf{A} \in \mathbb{R}^{n \times n}$ as the adjacency matrix of the graph, with elements $a_{ij} = 1 \iff (v_i, v_j) \in E$. With $\mathcal{N}(v)$ we denote the set of nodes adjacent to node v. Let also $\mathbf{D} \in \mathbb{R}^{n \times n}$ be the diagonal degree matrix where $d_{ii} = \sum_j a_{ij}$, and $\mathbf{L}$ the normalized graph laplacian defined by $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$, where $\mathbf{I}$ is the identity matrix.

With $GConv_\theta(\mathbf{x}_v, G)$ we denote a graph convolution with set of parameters θ. A GCN with k levels of convolutions is denoted as $GConv_{\theta_k}(\dots GConv_{\theta_1}(\mathbf{x}_v, G)\dots, G)$. For a discussion about the most common GCNs we refer to "Appendix A". We indicate with $\hat{\mathbf{X}}$ the input representation fed to a layer, where $\hat{\mathbf{X}} = \mathbf{X}$ if we are considering the first layer of the graph convolutional network, or $\hat{\mathbf{X}} = \mathbf{H}^{(i-1)}$ if considering the i-th graph convolution layer.

3 Background

The derivation of the graph convolution operator originates from graph spectral filtering (Defferrard et al. 2016). In order to set up a convolutional network on G, we need the notion of a convolution $*_G$ between a signal $\mathbf{x}$ and a filter signal $\mathbf{f}$ The Spectral convolution (Defferrard et al. 2016) can be considered the application of Fourier transform to graphs. It is obtained via Chebyshev polynomials of the Laplacian matrix. In general, the usage of a Chebyshev basis improves the stability in numerical approximation, and is defined as:

$$T^{(0)}(x) = 1, T^{(1)}(x) = x, T^{(k)}(x) = 2xT^{(k-1)}(x) - T^{(k-2)}(x).$$

A graph filter can be defined as:

$$\hat{\mathbf{F}}_\theta = \sum_{i=0}^{k^\star} \theta_i T^{(i)}(\tilde{\mathbf{\Lambda}}), \tag{1}$$

where $\tilde{\mathbf{\Lambda}} = \frac{2\mathbf{\Lambda}}{\lambda_{max}} - \mathbf{I}_n$ is the diagonal matrix of scaled eigenvectors of the graph Laplacian. The resulting convolution is then:

$$\mathbf{f}_\theta *_G \mathbf{x} = \sum_{i=0}^{k^\star} \theta_i T^{(i)}(\tilde{\mathbf{L}})\mathbf{x}, \tag{2}$$

where $\tilde{\mathbf{L}} = \frac{2\mathbf{L}}{\lambda_{max}} - \mathbf{I}_n$.

The graph convolution (Kipf and Welling 2017) (GCN) is a simplification of the spectral convolution. The authors propose to fix the order $k^\star = 1$ of the Chebyshev spectral convolution in Eq. (2) to obtain a linear first order graph convolution filter. These simple convolutions can then be stacked in order to improve the discriminatory power of the resulting network. The resulting convolution operator in Kipf and Welling (2017) is defined as:

$$\mathbf{H}^{(i+1)} = (\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}})\mathbf{H}^{(i)}\mathbf{W}^i, \tag{3}$$

where $\tilde{\mathbf{A}} = \mathbf{I_n} + \mathbf{A}$, $\mathbf{I_n}$ is the $n \times n$ identity matrix, $\tilde{\mathbf{D}}$ is a diagonal matrix with entries $\tilde{d}_{ii} = \sum_j \tilde{a}_{ij}$, and $\mathbf{H}^{(0)} = \mathbf{X}$.

Morris et al. (2019) defined the *GraphConv* operator inspired by the Weisfeiler-Lehman graph invariant, defined as follows:

$$\mathbf{H}^{(i+1)} = \mathbf{H}^{(i)}\bar{\mathbf{W}}^{(i)} + \mathbf{A}\mathbf{H}^{(i)}\hat{\mathbf{W}}^{(i)}. \tag{4}$$

Xu et al. (2018) defined the graph isomorphism network (GIN) convolution. The GIN convolution exploits aggregation over node neighbors is implemented using an MLP, therefore the resulting GC formulation is the following one:

$$\mathbf{H}^{(i+1)} = MLP((1+\epsilon)\mathbf{H}^{(i)} + \mathbf{A}\mathbf{H}^{(i)}). \tag{5}$$

4 Polynomial graph convolution (PGC)

In this section, we introduce the polynomial graph convolution (PGC), able to simultaneously and directly consider all topological receptive fields up to $k - hops$, just like the ones that are obtained by the graph convolutional layers in a stack of size k. PGC, however, does not incur in the typical limitation related to the complex interaction among the parameters of the GC layers. Actually, we show that PGC is *more* expressive than the most common convolution operators. Moreover, we prove that a single PGC convolution of order k is capable of implementing k linearly stacked layers of convolutions proposed in the literature, providing also *additional* functions that *cannot* be realized by the stack. Thus, the PGC layer extracts topological information from the input graph decoupling in an effective way the depth of the network from the size of the receptive field. Its combination with deep MLPs allows to obtain deep graph neural networks that can overcome the common oversmoothing problem of current architectures. The basic idea underpinning the definition of PGC is to consider the case in which the graph convolution can be expressed as a polynomial of the powers of a transformation $\mathcal{T}$ of the adjacency matrix. This definition is very general, and thus it incorporates many existing graph convolutions as special cases. Given a graph $G = (V, E, \mathbf{X})$ with adjacency matrix $\mathbf{A}$, the polynomial graph convolution (PGC) layer of degree k, transformation $\mathcal{T}$ of $\mathbf{A}$, and size m, is defined as

$$PGConv_{k,\mathcal{T},m}(\mathbf{X}, \mathbf{A}) = \mathbf{R}_{k,\mathcal{T}}\mathbf{W}, \tag{6}$$

where $\mathcal{T} : \bigcup_{j=1}^{\infty}(\mathbb{R}^{j\times j} \to \mathbb{R}^{j\times j})$ is a generic transformation of the adjacency matrix that preserves its shape, i.e. $\mathcal{T}(\mathbf{A}) \in \mathbb{R}^{n\times n}$. For instance, $\mathcal{T}$ can be defined as the function returning the Laplacian matrix starting from the adjacency matrix. Moreover, $\mathbf{R}_{k,\mathcal{T}} \in \mathbb{R}^{n\times s(k+1)}$, is defined as

$$\mathbf{R}_{k,\mathcal{T}} = [\mathbf{X}, \mathcal{T}(\mathbf{A})\mathbf{X}, \mathcal{T}(\mathbf{A})^2\mathbf{X}, .., \mathcal{T}(\mathbf{A})^k\mathbf{X}],$$

and $\mathbf{W} \in \mathbb{R}^{s(k+1)\times m}$ is a learnable weight matrix. For the sake of presentation, we will consider $\mathbf{W}$ as composed of blocks: $\mathbf{W} = [\mathbf{W}_0, \ldots, \mathbf{W}_k]^\top$, with $\mathbf{W}_j \in \mathbb{R}^{s\times m}$. In the following, we show that PGC is very expressive, able to implement commonly used convolutions as special cases.

4.1 Graph convolutions in literature as PGC instantiations

The PGC layer in (6) is designed to be a generalization of the linear stacking of some of the most common spatially localized graph convolutions. The idea is that spatially localized convolutions aggregate over neighbors (the message passing phase) using a transformation of the adjacency matrix (e.g. a normalized graph Laplacian). We provide

in this section a formal proof, as a theoretical contribution of this paper, that linearly stacked convolutions can be rewritten as polynomials of powers of the transformed adjacency matrix.

We start by showing how common graph convolution operators can be defined as particular instances of a single PGC layer (in most cases with $k = 1$). Then, we prove that linearly stacking any two PGC layers produces a convolution that can be written as a single PGC layer as well.

Spectral A layer of Spectral convolutions of order $k^\star$ defined in Eq. (2), can be implemented by a single PGC layer instantiating $\mathcal{T}(A)$ to be the graph Laplacian matrix (or one of its normalized versions), setting the PGC k value to $k^\star$, and setting the weight matrix to encode the constraints given by the Chebyshev polynomials. For instance, we can get the output $\mathbf{H}$ of a Spectral convolution layer with $k^\star = 3$ by the following PGC:

$$\mathbf{H} = [\mathbf{X}, \mathbf{L}\mathbf{X}, \mathbf{L}^2, \mathbf{L}^3\mathbf{X}]\mathbf{W}, \text{ where } \mathbf{W} = \begin{bmatrix} \mathbf{W}_0 - \mathbf{W}_2 \\ \mathbf{W}_1 - 3\mathbf{W}_3 \\ 2\mathbf{W}_2 \\ 4\mathbf{W}_3 \end{bmatrix}, \ \mathbf{W}_i \in \mathbb{R}^{s \times m}. \tag{7}$$

GCN The convolution defined in Eq. (3) can be obtained in the PGC framework by setting $k = 1$ and $\mathcal{T}(A) = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}} \in \mathbb{R}^{n \times n}$. We obtain the following equivalent equation:

$$\mathbf{H} = [\mathbf{X}, \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}]\mathbf{W}, \text{ where } \mathbf{W} = \begin{bmatrix} \mathbf{0} \\ \mathbf{W}_1 \end{bmatrix}, \tag{8}$$

where $\mathbf{0}$ is a $s \times m$ matrix with all entries equal to zero and $\mathbf{W}_1 \in \mathbb{R}^{s \times m}$ is the weight matrix of GCN. Note that the GCN does not consider a node differently from its neighbors, thus in this case there is no contribution from the first component of $\mathbf{R}_{k,\mathcal{T}}$.

GraphConv The convolution defined in Eq. (4) can be obtained by setting $\mathcal{T}(A) = A$ (the identity function), and k is again set to 1. A single *GraphConv* layer can be written as:

$$\mathbf{H} = [\mathbf{X}, \mathbf{A}\mathbf{X}]\mathbf{W}, \text{ where } \mathbf{W} = \begin{bmatrix} \mathbf{W}_0 \\ \mathbf{W}_1 \end{bmatrix}, \text{ and } \mathbf{W}_0, \mathbf{W}_1 \in \mathbb{R}^{s \times m}. \tag{9}$$

GIN

Technically, the GIN presented in eq (5) is a composition of a convolution (that is a linear operator) with a multi-layer perceptron. Let us thus decompose the *MLP*() as $f \circ g$, where g is an affine projection via weight matrix $\mathbf{W}$, and f incorporates the element-wise non-linearity, and the other layers of the MLP. We can thus isolate the GIN graph convolution component and define it as a specific PGC istantiation. We let $k = 1$ and $\mathcal{T}()$ the identity function as before. A single GIN layer can then be obtained as:

$$\mathbf{H} = [\mathbf{X}, \mathbf{A}\mathbf{X}]\mathbf{W}, \text{ where } \mathbf{W} = \begin{bmatrix} (1+\epsilon)\mathbf{W}_1 \\ \mathbf{W}_1 \end{bmatrix}. \tag{10}$$

Note that, differently from *GraphConv*, in this case the blocks of the matrix $\mathbf{W}$ are tied. Figure 3 in "Appendix B" depicts the expressivity of different graph convolution operators in terms of the respective constraints on the weight matrix $\mathbf{W}$. The comparison is made easy by the definition of the different graph convolution layers as instances of PGC layers. Actually, it is easy to see from Eqs. (8–10) that *GraphConv* is more expressive than GCN and GIN.

4.2 Linearly stacked graph convolutions as PGC instantiations

In the previous section, we have shown that common graph convolutions can be expressed as particular instantiations of a PGC layer. In this section, we show that a single PGC layer can model the linear stacking of any number of PGC layers (using the same transformation $\mathcal{T}$). Thus, a single PGC layer can model all the functions computed by arbitrarily many linearly stacked graph convolution layers defined in the previous section. We then show that a PGC layer includes also *additional* functions compared to the stacking of simpler PGC layers, which makes it more expressive.

Theorem 1 *Let us consider two linearly stacked PGC layers using the same transformation $\mathcal{T}$. The resulting linear Graph Convolutional network can be expressed by a single PGC layer.*

Due to space limitations, the proof is reported in "Appendix C". Here it is important to know that the proof of Theorem 1 tells us that a single PGC of order k can represent the linear stacking of any q ($\mathcal{T}$-compatible) convolutions such that $k = \sum_{i=1}^{q} d_i$, where d_i is the degree of convolution at level i. We will now show that a single PGC layer can represent also other functions, i.e. it is more general than the stacking of existing convolutions. Let us consider, for the sake of simplicity, the stacking of 2 PGC layers with $k = 1$ [that are equivalent to *GraphConv* layers, see Eq. (9)], each with parameters $\mathbf{W}^{(i)} = [\mathbf{W}_0^{(i)}, \mathbf{W}_1^{(i)}]^\top$, $i = 1, 2$, $\mathbf{W}_0^{(1)}, \mathbf{W}_1^{(1)} \in \mathbb{R}^{s \times m_1}$, $\mathbf{W}_0^{(2)}, \mathbf{W}_1^{(2)} \in \mathbb{R}^{m_1 \times m_2}$. The same reasoning can be applied to any other convolution among the ones presented in Sect. 4.1. We can explicitly write the equations computing the hidden representations:

$$\mathbf{H}^{(1)} = \mathbf{X}\mathbf{W}_0^{(1)} + \mathbf{A}\mathbf{X}\mathbf{W}_1^{(1)}, \tag{14}$$

$$\begin{aligned} \mathbf{H}^{(2)} &= \mathbf{H}^{(1)}\mathbf{W}_0^{(2)} + \mathbf{A}\mathbf{H}^{(1)}\mathbf{W}_1^{(2)} \\ &= \mathbf{X}\mathbf{W}_0^{(1)}\mathbf{W}_0^{(2)} + \mathbf{A}\mathbf{X}(\mathbf{W}_1^{(1)}\mathbf{W}_0^{(2)} + \mathbf{W}_0^{(1)}\mathbf{W}_1^{(2)}) + \mathbf{A}^2\mathbf{X}\mathbf{W}_1^{(1)}\mathbf{W}_1^{(2)}. \end{aligned} \tag{15}$$

A single PGC layer can implement this second order convolution as:

$$\mathbf{H}^{(2)} = [\mathbf{X}, \mathbf{A}\mathbf{X}, \mathbf{A}^2\mathbf{X}] \begin{bmatrix} \mathbf{W}_0^{(1)}\mathbf{W}_0^{(2)} \\ \mathbf{W}_1^{(1)}\mathbf{W}_0^{(2)} + \mathbf{W}_0^{(1)}\mathbf{W}_1^{(2)} \\ \mathbf{W}_1^{(1)}\mathbf{W}_1^{(2)} \end{bmatrix}. \tag{16}$$

Let us compare it with a PGC layer that corresponds to the same 2-layer architecture but that has no constraints on the weight matrix, i.e.:

$$\mathbf{H}^{(2)} = [\mathbf{X}, \mathbf{A}\mathbf{X}, \mathbf{A}^2\mathbf{X}] \begin{bmatrix} \mathbf{W}_0 \\ \mathbf{W}_1 \\ \mathbf{W}_2 \end{bmatrix}, \ \mathbf{W}_i \in \mathbb{R}^{s \times m_2}, \ i = 0, 1, 2. \tag{17}$$

Even though it is not obvious at a first glance, (16) is more constrained than (17), i.e. there are some values of $\mathbf{W}_0, \mathbf{W}_1, \mathbf{W}_2$ in (17) that cannot be obtained for any $\mathbf{W}^{(1)} = [\mathbf{W}_0^{(1)}, \mathbf{W}_1^{(1)}]^\top$ and $\mathbf{W}^{(2)} = [\mathbf{W}_0^{(2)}, \mathbf{W}_1^{(2)}]^\top$ in (16), as proven by the following theorem.

Theorem 2 *A PGC layer with $k = 2$ is more general than two stacked PGC layers with $k = 1$ with the same number of hidden units m.*

We refer the reader to "Appendix C" for the proof. Notice that the *GraphConv* layer is equivalent to a PGC layer with $k = 1$ (if no constraints on $\mathbf{W}$ are considered, see later in this section). Since the *GraphConv* is, in turn, more general than GCN and GIN, the above theorem holds also for those graph convolutions. Moreover, Theorem 2 trivially implies that a linear stack of q PGC layers with $k = 1$ is less expressive than a single PGC layer with $k = q$. In "Appendices D and E" we characterize the hypothesis that cannot be represented with the stacking approach, and we provide examples to provide more practical insights on such hypotheses and on the reason why they are not representable with stacking.

If we now consider that in many GCN architectures it is typical, and useful, to concatenate the output of all convolution layers before aggregating the node representations, then it is not difficult to see that such concatenation can be obtained by making wider the weight matrix of PGC. Let us thus consider a network that generates a hidden representation that is the concatenation of the different representations computed on each layer, i.e. $\mathbf{H} = [\mathbf{H}^{(1)}, \mathbf{H}^{(2)}] \in \mathbb{R}^{s \times m}$, $m = m_1 + m_2$. We can represent a two-layer *GraphConv* network as a single PGC layer as:

$$\mathbf{H} = [\mathbf{X}, \mathbf{AX}, \mathbf{A}^2\mathbf{X}] \begin{bmatrix} \mathbf{W}_0^{(1)} & \mathbf{W}_0^{(1)}\mathbf{W}_0^{(2)} \\ \mathbf{W}_1^{(1)} & \mathbf{W}_1^{(1)}\mathbf{W}_0^{(2)} + \mathbf{W}_0^{(1)}\mathbf{W}_1^{(2)} \\ \mathbf{0} & \mathbf{W}_1^{(1)}\mathbf{W}_1^{(2)} \end{bmatrix}. \tag{18}$$

More in general, if we consider k *GraphConv* convolutional layers (see Eq. (9)), each with parameters $\mathbf{W}^{(i)} = [\mathbf{W}_0^{(i)}, \mathbf{W}_1^{(i)}]^\top$, $i = 1, \dots, k$, $\mathbf{W}_0^{(i)}, \mathbf{W}_1^{(i)} \in \mathbb{R}^{m_{i-1} \times m_i}$, $m_0 = s$, $m = \sum_{j=1}^{k}$, the weight matrix $\mathbf{W} \in \mathbb{R}^{s \cdot (k+1) \times m}$ can be defined as follows:

$$\begin{bmatrix} F_{0,1}(\mathbf{W}^{(1)}) & F_{0,2}(\mathbf{W}^{(1)}, \mathbf{W}^{(2)}) & F_{0,3}(\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{W}^{(3)}) & \dots \\ F_{1,1}(\mathbf{W}^{(1)}) & F_{1,2}(\mathbf{W}^{(1)}, \mathbf{W}^{(2)}) & F_{1,3}(\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{W}^{(3)}) & \dots \\ \mathbf{0} & F_{2,2}(\mathbf{W}^{(1)}, \mathbf{W}^{(2)}) & F_{2,3}(\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{W}^{(3)}) & \dots \\ \mathbf{0} & \mathbf{0} & F_{3,3}(\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{W}^{(3)}) & \dots \\ \dots & \dots & \dots & \dots \end{bmatrix}, \tag{19}$$

where $F_{i,j}()$, $i,j \in \{0, \dots, k\}$, $i \leq j$, are defined as

$$F_{i,j}(\mathbf{W}^{(1)}, \dots, \mathbf{W}^{(j)}) = \sum_{\substack{(z_1, \dots z_j) \in \{0,1\}^j \\ s.t.\ \sum_{q=1}^{j} z_q = i}} \prod_{s=1}^{j} \mathbf{W}_{z_s}^{(s)}.$$

We can now generalize this formulation by concatenating the output of $k + 1$ PGC convolutions of degree ranging from 0 up to k. This gives rise to the following definitions:

$$\mathbf{W} = \begin{bmatrix} \mathbf{W}_{0,0} & \mathbf{W}_{0,1} & \mathbf{W}_{0,2} & \dots & \mathbf{W}_{0,k} \\ \mathbf{0} & \mathbf{W}_{1,1} & \mathbf{W}_{1,2} & \dots & \mathbf{W}_{1,k} \\ \mathbf{0} & \mathbf{0} & \mathbf{W}_{2,2} & \dots & \mathbf{W}_{2,k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{W}_{k,k} \end{bmatrix}, \ \mathbf{H} = \begin{bmatrix} (\mathbf{XW}_{0,0})^\top \\ (\mathbf{XW}_{0,1} + \mathcal{T}(\mathbf{A})\mathbf{XW}_{1,1})^\top \\ \vdots \\ (\mathbf{XW}_{0,k} + \dots + \mathcal{T}(\mathbf{A})^k\mathbf{XW}_{k,k})^\top \end{bmatrix}^\top, \tag{20}$$

where we do not put constraints among matrices $\mathbf{W}_{i,j} \in \mathbb{R}^{s \times m_j}$, $m = \sum_{j=0}^{k} m_j$, which are considered mutually independent. Note that as a consequence of Theorem 2, the network

defined in (20) is more expressive than the one obtained concatenating different *GraphConv* layers as defined in (19). It can also be noted that the same network can actually be seen as a single PGC layer of order $k+1$ with a constraint on the weight matrix (i.e., to be an upper triangular block matrix). Of course, any weights sharing policy can be easily implemented, e.g. by imposing $\forall j\ \mathbf{W}_{i,j} = \mathbf{W}_i$, which corresponds to the concatenation of the representations obtained at level i by a single stack of convolutions. In addition to reduce the number of free parameters, this weights sharing policy allows the reduction of the computational burden since the representation at level i is obtained by summing to the representation of level $i-1$ the contribution of matrix $\mathbf{W}_i$, i.e. $\mathbf{A}^i\mathbf{X}\mathbf{W}_i$

4.3 Computational complexity

As detailed in the previous discussion, the *degree* k of a PGC layer controls the size of its considered receptive field. In terms of the number of parameters, fixing the node attribute size s and the size m of the hidden representation, the number of parameters of the PGC is $O(s \cdot k \cdot m)$, i.e. it grows linearly in k. Thus, the number of parameters of a PGC layer is of the same order of magnitude compared to k stacked graph convolution layers based on message passing (Gilmer et al. 2017) (i.e. *GraphConv*, GIN and GCN, presented in Sect. 3).

If we consider the number of required matrix multiplications, compared to message passing GC networks, in our case it is possible to pre-compute the terms $\mathcal{T}(\mathbf{A})^i\mathbf{X}$ before the start of training, making the computational cost of the convolution calculation cheaper compared to message passing. In "Appendix H", we report an example that makes evident the significant improvement that can be gained in training time with respect to message passing.

5 Polynomial graph convolutional network (PGCN)

In this section, we present a neural architecture that exploits the PGC layer to perform graph classification tasks. Note that, differently from other GCN architectures, in our architecture (exploiting the PGC layer) the depth of the network is completely decoupled from the size k of the receptive field. The initial stage of the model consists of a first PGC layer with $k=1$. The role of this first layer is to develop an initial node embedding to help the subsequent PGC layer to fully exploit its power. In fact, in bioinformatics datasets where node labels $\mathbf{X}$ are one-hot encoded, all matrices $\mathbf{X}, \mathbf{A}\mathbf{X}, \ldots, \mathbf{A}^k\mathbf{X}$ are very sparse, which we observed, in preliminary experiments, influences in a negative way learning. Table 9 in "Appendix I" compares the sparsity of the PGC representation using the original one-hot encoded labels against their embedding obtained with the first PGC layer. The analysis shows that using this first layer the network can work on significantly denser representations of the nodes. Note that this first stage of the model does not significantly bound the PGC-layer expressiveness. A dense input for the PGC layer could have been obtained by using an embedding layer that is not a graph convolutional operator. However, this choice would have made difficult to compare our results with other state-of-the-art models in Sect. 7, since the same input transformation could have been applied to other models as well, making unclear the contribution of the PGC layer to the performance improvement. This is why we decided to use a PGC layer with $k=1$ (equivalent to a *GraphConv*) to compute the node embedding, making the results fully comparable since we are using

only graph convolutions in our PGCN. For what concerns the datasets that do not have the nodes' label (like the social networks datasets), using the PGC layer with $k = 1$ allows to create a label for each node that will be used by the subsequent larger PGC layer to compute richer node's representations. After this first PGC layer, a PGC layer as described in Eq. (20) of degree k is applied. In order to reduce the number of hyperparameters, we adopted the same number $\frac{m}{k+1}$ of columns (i.e., hidden units) for matrices $\mathbf{W}_{i,j}$, i.e. $\mathbf{W}_{i,j} \in \mathbb{R}^{s \times \frac{m}{k+1}}$. A graph-level representation $\mathbf{s} \in \mathbb{R}^{m*3}$ based on the PGC layer output $\mathbf{H}$ is obtained by an aggregation layer that exploits three different aggregation strategies over the whole set of nodes V, $j = 1, \dots, m$:

$$s_j^{avg} = avg(\{h_v^{(j)} | v \in V\}),\ s_j^{max} = max(\{h_v^{(j)} | v \in V\}),\ s_j^{sum} = sum(\{h_v^{(j)} | v \in V\}),$$
$$\mathbf{s} = [s_1^{avg}, s_1^{max}, s_1^{sum}, \dots, s_m^{avg}, s_m^{max}, s_m^{sum}]^\top.$$

The readout part of the model is composed of q dense feed-forward layers, where we consider q and the number of neurons per layer as hyper-parameters. Each one of these layers uses the *ReLU* activation function, and is defined as $\mathbf{y}_j = ReLu(\mathbf{W}_j^{readout}\mathbf{y}_{j-1} + \mathbf{b}^{readout})$, $j = 1, \dots, q$, where $\mathbf{y}_0 = \mathbf{s}$. Finally, the output layer of the PGCN for a c-class classification task is defined as:

$$\mathbf{o} = LogSoftmax(\mathbf{W}^{out}\mathbf{y}_q + \mathbf{b}^{out}).$$

For a complete discussion about the reasons why we implement the readout network by an MLP, please refer to "Appendix J".

6 Multi-scale GCN architectures in literature

Some recent works in literature exploit the idea of extending graph convolution layers to increase the receptive field size. In general, the majority of these models, that concatenate polynomial powers of the adjacency matrix A, are designed to perform node classification, while the proposed PGCN is developed to perform graph classification. In this regard, we want to point out that the novelty introduced in this paper is not limited to a novel GC-layer, but the proposed PGCN is a complete architecture to perform graph classification.

Atwood and Towsley (2016) proposed a method that exploits the power series of the probability transition matrix, that is multiplied (using Hadamard product) by the inputs. The method differs from the PGCN both in terms of how the activation is computed, and because the activation computed for each exponentiation is summed, instead been concatenated.

Similarly in Defferrard et al. (2016) the model exploits the Chebyshev polynomials, and, differently from PGCN, it sums them over k. This architectural choice makes the proposed method less general than the PGCN. Indeed, as showed in Sect. 4.1, the model proposed in Defferrard et al. (2016) is an instance of the PGC.

In Xu et al. (2018), the authors proposed to modify the common aggregation layer in such a way that, for each node, the model aggregates all the intermediate representations computed in the previous GC-layers. In this work, differently from PGCN, the model exploits the message passing method introducing a bias in the flow of the topological information. Note that, as proven in Theorem 2, a PGC layer of degree k is not equivalent to concatenating the output of k stacked GC layers, even though the PGC layer can also implement this particular architecture.

Another interesting approach is proposed in Tran et al. (2018), where the authors consider larger receptive fields compared to standard graph convolutions. However, they focus on a single convolution definition (using just the adjacency matrix) and consider shortest paths (differently from PGCN that exploits matrix exponentiations, i.e. random walks). In terms of expressiveness, it is complex to compare methods that exploit matrix exponentiations with methods based on the shortest paths. However, it is interesting to notice that, thanks to the very general structure of the PGC layer, it is easy to modify the PGC definition in order to use the shortest paths instead of the adjacency matrix transformation exponentiations. In particular, we plan to explore this option as the future development of the PGCN.

Wu et al. (2019) introduce a simplification of the graph convolution operator, dubbed simple graph convolution (SGC). The model proposed is based on the idea that perhaps the nonlinear operator introduced by GCNs is not essential, and basically, the authors propose to stack several linear GC operators. In Theorem 2 we prove that staking k GC layers is less expressive than using a single PGC layer of degree k. Therefore, we can conclude that the PGC Layer is a generalization of the SGC.

In Liao et al. (2019) the authors construct a deep graph convolutional network, exploiting particular localized polynomial filters based on the Lanczos algorithm, which leverages multi-scale information. This convolution can be easily implemented by a PGC layer. In Chen et al. (2019) the authors propose to replace the neighbor aggregation function with graph augmented features. These graph augmented features combine node degree features and multi-scale graph propagated features. Basically, the proposed model concatenates the node degree with the power series of the normalized adjacency matrix. Note that the graph augmented features differ from $\mathbf{R}_{k,\mathcal{T}}$, used in the PGC layer. Another difference with respect to the PGCN resides on the subsequent part of the model. Indeed, instead of projecting the multi-scale features layer using a structured weights matrix, the model proposed in Chen et al. (2019) aggregates the graph augmented features of each vertex and project each of these subsets by using an MLP. The model readout then sums the obtained results over all vertices and projects it using another MLP.

Luan et al. (2019) introduced two deep GCNs that rely on Krylov blocks. The first one exploits a GC layer, named snowball, that concatenates multi-scale features incrementally, resulting in a densely-connected graph network. The architecture stacks several layers and exploits nonlinear activation functions. Both these aspects make the gradient flow more complex compared to the PGCN. The second model, called Truncated Krylov, concatenates multi-scale features in each layer. In this model, differently from PGCN, the weights matrix of each layer has no structure, thus topological features from all levels are mixed together.

Another method that introduces an alternative to the message passing mechanism is proposed in Klicpera et al. (2019). Differently from PGCN, that exploits the concatenation of the powers of the diffusion operator to propagate the message thought the graph topology, Klicpera et al. proposed a graph convolution that exploits the Personalized PageRank as propagation schema. Let $f(\cdot)$ define a 2-layer feedforward neural network. The PPNP layer is defined as: $\mathbf{H} = \alpha\left(\mathbf{I}_n - (1-\alpha)\tilde{\mathbf{A}}\right)^{-1} f(\mathbf{X})$, where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_n$. Such filter preserves locality due to the properties of Personalized PageRank.

The same paper proposed an approximation, derived by a truncated power iteration, avoiding the expensive computation of the matrix inversion, referred to as APPNP. It is implemented as a multi-layer network where the $(l+1)$-th layer is defined as $\mathbf{H}^{(l+1)} = (1-\alpha)\tilde{\mathbf{S}}\mathbf{H}^{(l)} + \alpha\mathbf{H}^{(0)}$, where $\mathbf{H}^{(0)} = f(\mathbf{X})$ and $\tilde{\mathbf{S}}$ is the renormalized adjacency matrix adopted in GCN, i.e. $\tilde{\mathbf{S}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$.

PPNP and APPNP differ significantly from PGCN, since they use a multi-layer (non-convolutional) architecture in order to exploit the different powers of the diffusion operator. Other important differences between PGCN and PPNP/APPNP, is that the second ones are specifically developed to solve the node classification task.

Abu-El-Haija et al. (2019) proposed a multilayer architecture that exploits the MixHop Graph Convolution Layer. Each layer of the model mixes a subset (managed as a hyper-parameter) of the powers of the adjacency matrix, by multiplying them by the embedding computed in the previous layer. Finally, each layer concatenates the representation obtained for each considered diffusion operator's powers. Therefore, differently from PGC, the MixHop layer considers a subset of the powers of the adjacency matrix, and moreover, it non-linearly projects the representation obtained for each considered power. Similarly to the previously discussed multi-scale architectures, also MixHop model is developed to perform the node classification task.

Rossi et al. (2020) proposed an alternative method, named SIGN, to scale GNN to a very large graph. This method uses as a building block the set of exponentiations of linear diffusion operators. In this building block, every exponentiation of the diffusion operator is linearly projected by a learnable matrix. Moreover, differently from the PGC layer, a non-linear function is applied on the concatenation of the diffusion operators making the gradient flow more complex compared to the PGCN.

Very recently Liu et al. (2020) proposed a model dubbed deep adaptive graph neural network, to learn node representations by adaptively incorporating information from large receptive fields. Differently from PGCN, first, the model exploits an MLP network for node feature transformation. Then it constructs a multi-scale representation leveraging on the computed nodes features transformation and the exponentiation of the adjacency matrix. This representation is obtained by stacking the various adjacency matrix exponentiations (thus obtaining a 3-dimensional tensor). Similarly to Luan et al. (2019), also in this case the model projects the obtained multi-scale representation using weights matrix that has no structure, obtaining that the topological features from all levels are mixed together. Moreover, this projection uses also (trainable) retainment scores. These scores measure how much information of the corresponding representations derived by different propagation layers should be retained to generate the final representation for each node in order to adaptively balance the information from local and global neighborhoods. Obviously, that makes the gradient flow more complex compared to the PGCN, and also impact the computational complexity.

7 Experimental setup and results

In this section, we introduce our model set up, the adopted datasets, the baselines models, and the hyper-parameters selection strategy. We then report and discuss the results obtained by the PGCN. For implementation details please refer to Appendix F.

7.1 Dataset

We empirically validated the proposed PGCN on five commonly adopted graph classification benchmarks modeling bioinformatics problems: PTC (Helma et al. 2001), NCI1 (Wale et al. 2008), PROTEINS (Borgwardt et al. 2005), D&D (Dobson and Doig 2003) and ENZYMES (Borgwardt et al. 2005). Moreover, we also evaluated the PGCN on 3

large graph social datasets: COLLAB, IMDB-B, IMDB-M (Yanardag and Vishwanathan 2015). We report more details in "Appendix G".

7.2 Baselines and hyper-parameter selection

We compare PGCN versus several GNN architectures, that achieved state-of-the-art results on the used datasets. Specifically, we considered PSCN (Niepert et al. 2016), Funnel GCNN (FGCNN) model (Navarin et al. 2020), DGCNN (Zhang et al. 2018), GIN (Xu et al. 2019), DIFFPOOL (Ying et al. 2018) and GraphSage (Hamilton et al. 2017). Note that these graph classification models exploit the convolutions presented in Sect. 3. We report also the results of a baseline model that is structure-agnostic from Errica et al. (2020). More precisely in Errica et al. (2020), the authors adopted two different baselines, one for the chemical datasets and one for social datasets. For the chemical datasets, the model counts the occurrences of atom types in the graph by summing the features of all nodes in the graph together and then applies a single layer MLP. For social datasets, the baseline the model applies an MLP that has in input the nodes' features then uses a global sum pooling operator and then another MLP to perform classification.

The results were obtained by performing five runs of ten-fold cross-validation. The hyper-parameters of the model (number of hidden units, learning rate, weight decay, k, q) were selected using a grid search, where the explored sets of values were changed based on the considered dataset. As validation test methodology we decided to follow the method proposed in Errica et al. (2020), that in our opinion, turns out to be the fairest. Other details about validation are reported in "Appendix K".

7.3 Results and discussion

The results reported in Table 1 show that the PGCN achieves higher results in all (except one) considered datasets compared to competing methods. In particular, on NCI1 and ENZYMES the proposed method outperforms state-of-the-art results. In fact, in both cases, the performances of PGCN and the best competing method are more than one standard deviation apart. Even for PTC, D&D, PROTEINS, IMDB-B and IMDB-M datasets PGCN shows a slight improvement over the results of FGCNN and DGCNN models. Furthermore, the results of PGCN in Bioinformatics datasets achieves a significant lower standard deviation (evaluated over the 5 runs of 10-fold cross-validation). For what concerns the COLLAB datasets, PGCN obtained the second higher result in the considered state-of-the-art methods. Note however that the difference with respect to the first one (GIN) is within the standard deviation.

Significativity of our results To understand if the improvements reported in Table 1 are significant or can be attributed to random chance, we conducted the two-tailed Wilcoxon Signed-Ranks test between our proposed PGCN and competing methods. This test considers all the results for the different datasets at the same time. According to this test, PGCN performs significantly better (p-value < 0.05) than PSCN, DGCNN[3], GIN, DIFFPOOL and GraphSAGE. As for FGCNN and DGCNN[2], four datasets are not enough to conduct the Wilcoxon test, see Japkowicz and Shah (2011), Table A.5.

Table 1 Accuracy comparison between PGCNN and several *state-of-the-art* models on graph classification task.

Model \ Dataset	PTC	NCI1	PROTEINS	D&D	ENZYMES	COLLAB	IMDB-B	IMDB-M
PSCN[1]	60.00	76.34	75.00	76.27	-	72.60	71.00	45.23
	±4.82	±1.68	±2.51	±2.64	-	±2.15	±2.29	±2.84
FGCNN[2]	58.82	81.50	74.57	77.47	-	-	-	-
	±1.80	±0.39	±0.80	±0.86	-	-	-	-
DGCNN[2]	57.14	72.97	73.96	78.09	-	-	-	-
	±2.19	±0.87	±0.41	±0.72	-	-	-	-
DGCNN[3]	-	76.4	72.9	76.6	38.9	57.4	53.3	38.6
	-	±1.7	±3.5	±4.3	±5.7	±1.9	±5.0	±2.2
GIN[3]	-	80.0	73.3	75.3	59.6	**75.9**	66.8	42.2
	-	±1.4	±4.0	±2.9	±4.5	**±1.9**	±3.9	±4.6
DIFFPOOL[3]	-	76.9	73.7	75.0	59.5	67.7	68.3	45.1
	-	±1.9	±3.5	±3.5	±5.6	±1.9	±6.1	±3.2
GraphSAGE[3]	-	76.0	73.0	72.9	58.2	71.6	69.9	47.2
	-	±1.8	±4.5	±2.0	±6.0	±1.5	±4.6	±3.6
Baseline[3]	-	69.8	**75.8**	78.4	65.2	55.0	50.7	36.1
	-	±2.2	**±3.7**	±4.5	±6.4	±1.9	±2.4	±3.0
PGCN	**60.50**	**82.04**	75.31	**79.45**	**70.5**	74.1	**72.60**	**47.39**
	±0.67	**±0.26**	±0.31	**±0.29**	**±1.77**	±1.69	**±3.80**	**±3.51**

The best results are highlighted in bold.

[1]Niepert et al. (2016) , [2]Navarin et al. (2020), [3]Errica et al. (2020).

7.4 Experimental results omitted in the results comparison

The results reported in Xu et al. (2019), Chen et al. (2019), Ying et al. (2018) are not considered in our comparison since the model selection strategy is different from the one we adopted and this makes the results not comparable.

The importance of the validation strategy is discussed in Errica et al. (2020), where results of a fair comparison among the considered baseline models are reported. For the sake of completeness, we also report (and compare) in Table 2 the results obtained by evaluating the PGCN method with the validation policy used in Xu et al. (2019), Chen et al. (2019), Ying et al. (2018).

Specifically, the results reported in Xu et al. (2019), Chen et al. (2019), Ying et al. (2018) are not considered in our experimental comparison since the model selection strategy is different from the one we adopted. Indeed the results reported cannot be compared with the other results reported in Table 1 of the paper, because the authors state "*The hyper-parameters we tune for each dataset are [...] the number of epochs, i.e., a single epoch with the best cross-validation accuracy averaged over the 10 folds was selected.*". Similarly, for the result reported in Chen et al. (2019) for the GCN and the GFN models, the authors state "*We run the model for 100 epochs, and select the epoch in the same way as* Xu et al. (2019), i.e., a single epoch with the best cross-validation accuracy averaged over the ten folds is selected". In both cases, the model selection strategy is clearly biased and different from the one we adopted. This makes the results not comparable.

Table 2 PGCN accuracy comparison using different values of k.

Model \ dataset	PTC	NCI1	PROTEINS	D&D	ENZYMES	COLLAB	IMDB-B	IMDB-M
GIN[a]	64.6	82.7	76.2	-	-	80.2	75.1	52.3
	±7.0	±1.7	±2.8	-	-	±1.9	±5.1	±2.8
GFN[b]	-	82.77	76.46	78.78	70.17	81.50	73.00	51.80
	-	±1.49	±4.06	±3.49	±5.58	±2.42	±4.35	±5.16
GCN[b]	-	83.65	75.65	79.12	69.50	**81.72**	73.30	51.20
	-	±1.69	±3.24	±3.07	±7.37	**±1.64**	±5.29	±5.13
DIFFPOOL[c]	-	-	76.25	80.64	62.53	75.48	-	-
PGCN	**74.0**	**84.40**	**79.20**	**82.54**	**78.17**	76.96	**77.88**	**52.97**
	±2.81	**±0.14**	**±0.81**	**±1.09**	**±0.68**	±2.14	**±3.11**	**±2.67**

The validation policy is the same used in Xu et al. (2019), Chen et al. (2019) and Ying et al. (2018). In Ying et al. (2018) the variance is not reported. The best results are highlighted in bold.

[a] (Xu et al. 2019).

[b] (Chen et al. 2019).

[c] (Ying et al. 2018).

Moreover, in Xu et al. (2019) the node descriptors are augmented with structural features. In GIN experiments the authors add a one-hot representation of the node degree. We decided to use a common setting for the chemical domain, where the nodes are labeled with a one-hot encoding of their atom type. The only exception is ENZYMES, where it is common to use 18 additional available features. Also in Ying et al. (2018) there is a similar problem since the authors add the degree and the clustering coefficient to each node feature vector. For the sake of completeness in Table 2 we report the results obtained by the proposed method following the same validation policy used in Xu et al. (2019), Chen et al. (2019), Ying et al. (2018). The table shows that the PGCN outperforms the methods proposed in the literature in almost all datasets.

7.5 Empirical comparison versus multi-scale GCNs

In this section, we empirically compare the PGCN with some of the methods that exploit the multi-scale approach. As we highlight in Section 6, most of these models are developed to perform node classification tasks and have very specific implementations. That makes their extension to graph classification task very complex, and in some cases, it drives to a modification that defines an almost completely brand new model. For these reasons, we decided to implement only the models where the multi-scale layer, or at least the proposed mechanism, is already implemented in *pytorch geometric*. Thanks to this, we were able to adapt these models by inserting after the multi-scale layer representation the same readout structure used by the PGCN. The models for which we developed and implemented a graph classification version (never proposed in the literature) are four, i.e. the SGC Wu et al. (2019), Chebyshev Convolutional network (Defferrard et al. 2016), JK-Net (Xu et al. 2018), and SIGN (Rossi et al. 2020). For what concerns the JK-net, the authors propose a novel methodology to aggregate the different convolutional layer representations. Therefore, in order to perform a fair comparison, we consider the parameter k as the number of convolutional layers of the model. Moreover, as graph convolutional operator we exploited the GCN (Kipf and Welling 2017), as did by the authors in the original paper. For all these

Table 3 PGCN accuracy comparison versus other multi-scale node methods adapted to perform graph classification tasks

Model \ Dataset	PTC	NCI1	PROTEINS	D&D	ENZYMES	COLLAB	IMDB-B	IMDB-M
SGC	55.63	76.25	75.43	77.10	31.33	69.34	66.38	43.33
	±7.64	±2.51	±3.43	±4.43	±5.61	±1.70	±5.46	±3.35
Cheb-Net	55.22	80.91	**75.80**	77.89	38.13	**74.36**	70.62	43.90
	±6.54	±1.85	±5.12	±3.67	±6.20	**±1.97**	±3.82	±3.42
JK-Net	57.58	76.79	74.09	77.42	31.16	73.56	64.80	38.84
	±6.95	±2.34	±3.54	±3.16	±6.80	±2.10	±4.21	±3.78
SIGN	54.98	77.10	75.54	77.19	42.75	58.47	63.68	39.63
	±8.34	±2.47	±4.35	±3.85	±6.51	±3.73	±4.61	±3.79
PGCN	**60.50**	**82.04**	75.31	**79.45**	**70.5**	74.1	**72.60**	**47.39**
	±0.67	**±0.26**	±0.31	**±0.29**	**±1.77**	±1.69	**±3.80**	**±3.51**

The validation policy is the same as the one used for the results reported in Errica et al. (2020). The best results are highlighted in bold.

models we performed a full validation by exploiting the GRID-search following the same methodology used to validate the PGCN results, and using the same hyper-parameters grid (reported in Appendix K). Performing the validation via GRID search, for all the considered datasets, has required to run more than 55, 000 experiments.[1] Note that also the model proposed by Klicpera et al. (2019) is present as a propagation mechanism in *pytorch geometric* library, but it cannot be applied on graph classification task since PNPP is specifically defined for node classification. Finally, we would like to point out that the only models cited in Section 6 that are already defined to perform graph classification are the ones proposed in Chen et al. (2019), and considered in the comparison reported in Table 2.

The obtained results are reported in Table 3, and they show that the PGCN model achieves better results than the other multi-scale architectures on all the considered datasets, except PROTEINS and COLLAB. Notice that the difference between the accuracy reached by PGCN and the best result achieved on the PROTEINS and COLLAB datasets by the Chebyshev Convolutional network, is significantly lower than the standard deviation obtained by the best method. For what concerns the bio-informatic datasets, the PGCN consistently shows a lower standard deviation in comparison with the other multi-scale models.

Similarly to what done in Sect. 7, in order to prove that the improvements reported in Table 3 are significant, and not due to chance, we conducted the two-tailed Wilcoxon Signed-Ranks test between our proposed PGCN and multi-scale competing methods. Also in this case, the PGCN improvement compared to all competing methods is statistically significant (significance level < 0.05).

[1] The code used to perform the experiments is publicly available at the following URL: https://github.com/lpasa/MultiScale_GCNs

Table 4 PGCN accuracy comparison on the validation set of the datasets for different values of k.

Dataset \ k	2	3	4	5
PTC	**74.0**	68.86	69.14	69.43
	±2.81	±2.44	±2.33	±1.78
NCI1	82.68	83.16	**84.40**	84.04
	±0.22	±0.70	**±0.14**	±0.83
PROTEINS	**79.20**	78.48	78.48	78.84
	±0.81	±0.56	±1.04	±0.80
D&D	81.95	82.03	81.69	**82.54**
	±1.19	±1.06	±0.68	**±1.09**
ENZYMES	77.50	76.83	**78.17**	77.18
	±1.27	±0.85	**±0.68**	±0.92
Dataset \ k	2	4	6	8
COLLAB	76.94	76.32	**76.96**	76.72
	±1.47	±1.45	**±2.14**	±1.60
IMDB-B	76.70	76.60	76.38	**77.88**
	±3.48	±3.84	±2.90	**±3.11**
IMDB-M	52.37	52.7	52.41	**52.97**
	±3.02	±2.84	±2.85	**±2.67**

The best results are highlighted in bold.

8 Model analysis

In this section, we analyze some crucial aspects of the proposed model. Specifically, we studied the impact of the size of the receptive fields, and the computation demand of the model, comparing the time performance of the proposed model vs FGCNN (Navarin et al. 2020), that shares a similar architecture.

8.1 Impact of receptive field size on PGCN

Most of the proposed GCN architectures in literature generally stack 4 or fewer GCs layers. The proposed PGC layer allows us to represent a linear version of these architectures by using a single layer with an even higher depth ($k + 1$), without incurring in problems related to the flow of the topological information. Different values of k have been tested to study how much the capability of the model to represent increased topological information helps to obtain better results. The results of these experiments are reported in Table 4. The accuracy results in this table are referred to the *validation* sets, since the choice of k is part of the model selection procedure. We decided to take into account a range of k values between 2 and 5 for bioinformatics datasets, and between 2 to 8 for social networks datasets. The results show that it is crucial to select an appropriate value for k. Several factors influence how much depth is needed. It is important to take into account that the various datasets used for the experiments refer to different tasks. The quantity and the type of topological information required (or useful) to solve the task highly influences the choice of k. Moreover, also the input dimensions and the

Table 5 Time in second to perform a single training epoch (2*nd* and 3*rd* column) and to perform classification (4*th* and 5*th* column), using PGCN and FGCNN (Navarin et al. 2020), respectively.

Dataset \ Model	Train		Classification	
	PGCN	FGCNN	PGCN	FGCNN
D&D	0.718±0.098	0.975±0.146	0.054±0.011	0.055±0.006
ENZYMES	0.164±0.015	0.247±0.032	0.011±0.001	0.016±0.002
NCI1	0.883±0.119	1.568±0.263	0.052±0.005	0.089±0.011
PROTEINS	0.296±0.036	0.456±0.0599	0.024±0.003	0.027±0.004
PTC	0.084±0.009	0.139±0.016	0.006±0.002	0.009±0.003
COLLAB	1.507±0.175	2.048±0.378	0.137±0.014	0.109±0.014
IMDB-B	0.223±0.024	0.373±0.054	0.018±0.003	0.027±0.004
IMDB-M	0.326±0.044	0.554±0.087	0.022±0.003	0.034±0.005

number of graphs contained in a dataset play an important role. In fact, using higher values of k increases the number columns of the $\mathbf{R}_{k,\mathcal{T}}$ matrix (and therefore the number of parameters embedded in $\mathbf{W}$), making the training of the model more difficult. It is interesting to notice that in many cases our method exploits a larger receptive field (i.e. a higher *degree*) compared to the competing models. Note that the datasets where better results are obtained with $k = 2$ (PTC and PROTEINS) contain a limited amount of training samples, thus deeper models tend to overfit arguably for the limited amount of training data.

8.2 Speed of convergence

Here, we discuss the results in terms of computation demand between a proposed PGCN and FGCNN (Navarin et al. 2020). We decided to compare these two models since they present a similar readout layer, therefore the comparison best highlights how the different methodology manage the number of considered k-hops, from the point of view of performance. In Table 5, we report the average time (over the ten folds) to perform a single epoch of training and to perform the classification with both method.

In the evaluation we considered similar architectures using 3 layers for the FGCNN and $k = 2$ for PGCN. The other hyper-parameters were set with the aim to get almost the same number of parameters in both models, to ensure a fair comparison. The batch sizes used for this evaluation are the same selected by the PGCN model selection. The results show a significant advantage in using a PGC layer instead of the message passing based method exploited by FCGNN.

Concerning the speed of convergence of the two models, in Fig. 1 we report the training curves for two representative datasets (D&D and NCI). In the x-axis we report the computational time in seconds, while in the y-axis we report the loss value. Both curves end after 200 training epochs. From the curves it can be seen that PGCN converges faster or with a similar pace than FCGNN.

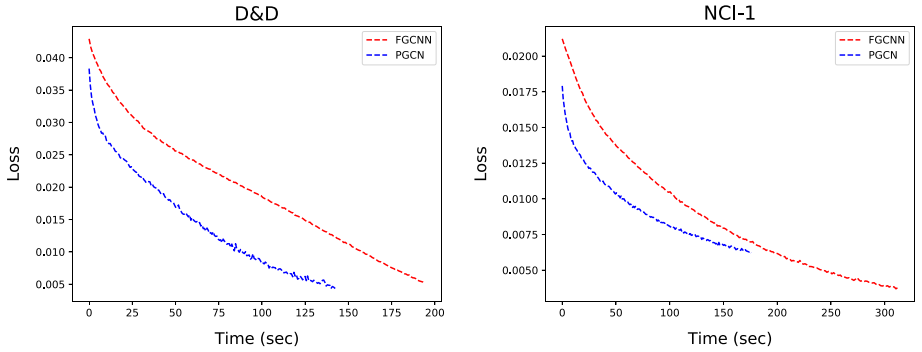

Fig. 1 PGCN and FGCNN training curves for D&D and NCI1 datasets

8.3 W structure

PGC has been defined with a $\mathbf{W}$ matrix that is a block upper triangular matrix. This structure is induced by the concatenation of the output of k PGC convolutions of degree ranging from 0 up to k. Indeed, using a matrix with this precise structure ensures us to obtain multi-resolution representations for each node and its neighborhoods. In fact, $PGConv_{i,\mathcal{T},m}(\mathbf{X}, \mathbf{A})$, $i \in [0, \dots, k]$ contains information only about random walks of length exactly equal to i. Thanks to the structure of $\mathbf{W}$, different hop components are progressively mixed in a differentiated way. But this is not the only advantage obtained by using this particular structure: having a block triangular matrix increases the probability to keep a full rank matrix during training, thus to fully exploit the expressivity of the corresponding linear transformation. These two features can also be obtained by imposing even more strict constraints on W structure, further reducing the number of used parameters. For example, if we use a block diagonal matrix, we have that the node embeddings computed for each i value will not be mixed when computing the corresponding hidden representations $\mathbf{H}$. An even more constrained structure for W can be obtained by using a diagonal matrix. In this case, not only the embeddings for different i values will not be mixed, but also the single node features will not be mixed each other when computing $\mathbf{H}$. Note that, for both these matrix structures the number of trainable parameters decreases significantly with respect to the block upper triangular case.

In Fig. 2, we report the (average) distribution of the singular values of the weights $\mathbf{W}$ along with the heatmaps that show the structure of the matrix and the (average) values of the weights. The considered weights matrices are the results of the training phase on the NCI1 dataset using the same hyper-parameters. The singular values are computed for each W matrix learned in each of the 10-fold used to compute the reported accuracy. In the figure, we report the average (and variance) of each singular value of all folds. The obtained curves show that using a more sparse structure the number of singular values close to 0 decreases. As we pointed out, having a W with block diagonal structure, or even a diagonal structure, limits the expressiveness of the computed transformation, but on the other hand, it allows to have a faster training phase (due to the lower number of trainable parameters). To assess the pros and cons of using these more constrained structures, we report in Table 6 the accuracy obtained by the same model using each of the proposed $\mathbf{W}$ structures

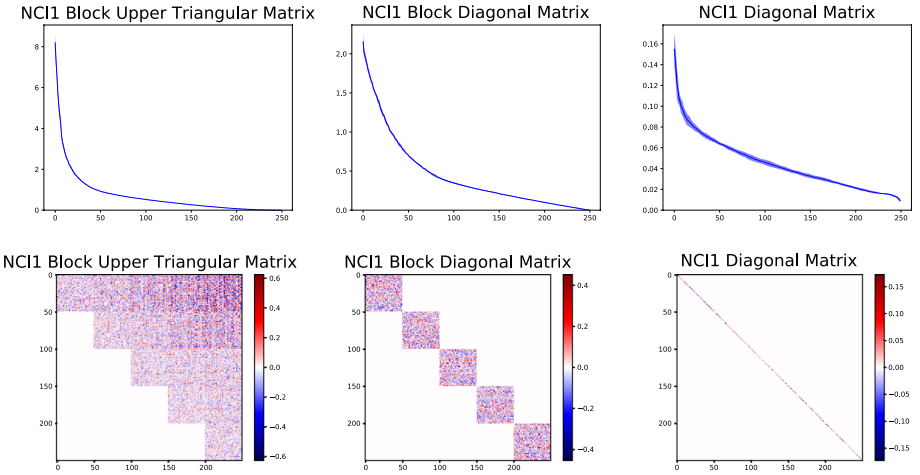

Fig. 2 Top: singular value distributions (average with variance) of the weights matrices **W** for different matrix structures (block upper triangular, block diagonal, diagonal) for the NCI1 dataset (10-folds). Bottom: heatmaps of the corresponding **W** matrices (average over the 10-fold)

Table 6 Accuracy computed on NCI1 test set varying the structure of the W matrix.

W structure	m	#Parameter	Accuracy
Block upper triangular matrix	50	135627	82.04±0.26
Block diagonal matrix	50	110627	81.09±1.85
Diagonal matrix	50	98377	80.41±2.21

for the NCI1 dataset. The results show that using a more sparse structure the performance drops, but not in a substantial way, still maintaining a good classification accuracy.

9 Discussion about further PGC advantages

In this section, we briefly discuss some additional advantages that the proposed PGC architecture does possess. In a nutshell, there are two main additional advantages: *i)* the architecture is amenable to the application of techniques to improve explainability; *ii)* singular values of **W** can ease the model selection procedure. Concerning explainability, if a diagonal **W** is used, any sensitivity/relevance analysis applied to the final network with respect to the input will allow to understand which specific node feature[2] is important at which specific value of k. This information can be projected back to a specific input graph, giving the possibility to explain what are the (node and structural) input features that most contribute to the output. If a block diagonal **W** is used, features are mixed, but still the most important values of k (structural features) for the output task can be easily identified. Concerning model selection, the singular values of **W** provide a guidance on the sizing of the network. In fact, independently from the imposed structure, too low singular values for **W** are an

[2] Assuming no node embeddings are used.

indication that smaller sizes for it may be preferred if forcing those singular values to 0 (easily achievable by a Singular Value Decomposition of $\mathbf{W}$) returns the same (or better) performances. Of course, using a (block) diagonal matrix will make this process much easier.

10 Conclusions and Future Works

In this paper, we analyze some of the most common convolution operators evaluating their expressiveness. Our study shows that their linear composition can be defined as instances of a more general Polynomial Graph Convolution operator with a higher expressiveness. We defined an architecture exploiting a single PGC layer to generate a decoupled representation for each neighbor node at a different topological distance. This strategy allows us to avoid the bias on the flow of topological information introduced by stacking multiple graph convolution layers. We empirically validated the proposed Polynomial Graph Convolutional Network on five commonly adopted graph classification benchmarks. The results show that the proposed model outperforms competing methods in almost all the considered datasets, showing also a more stable behavior.

In the future, in addition to exploring the research directions drafted in Sect. 9, we plan to study the possibility to introduce an *attention mechanism* by learning a transformation $\mathcal{T}$ that can adapt to the input. Furthermore, we will explore whether adopting our PGC operator as a large random projection can allow to develop a novel model for learning on graph domains.

Appendix A: Graph neural networks

A graph neural network (GNN) is a neural network model that exploits the structure of the graph and the information embedded in feature vectors of each node to learn a classifier or regressor on a graph domain. Due to the success in image processing, convolutional-based neural networks have become one of the main architectures (ConvGNN) applied to graph processing. The typical structure of a ConvGNN comprises a first part of processing where convolutional layers are used in order to learn a representation $\mathbf{h}_v \in \mathbb{R}^m$ for each vertex $v \in V$. These representations are then *combined* to get a representation of the whole graph, so that a standard feed-forward (deep) neural network can be used to process it. Convolutional layers are important since they define *how* the (local) topological information is mixed with the information attached to involved nodes, and *what* is the information to pass over to the subsequent computational layers. Because of that, several convolution operators for graphs have recently been proposed (Defferrard et al. 2016; Kipf and Welling 2017; Morris et al. 2019; Xu et al. 2019).

The first definition of neural network for graphs has been proposed by Sperduti and Starita (1997). More recently, (Micheli 2009) proposed the Neural Network for Graphs (NN4G), exploiting an idea that has been re-branded later as *graph convolution*, and (Scarselli et al. 2008) defined a recurrent neural network for graphs. In the last few years, several models inspired by the graph convolution have been proposed. Many recent works defining graph convolutional networks (GCNs) extend the NN4G formulation (Micheli 2009), for instance the graph convolutional network (GCN) (Kipf and Welling 2017) is based on a linear first-order filter based on the normalized graph Laplacian for each graph convolutional

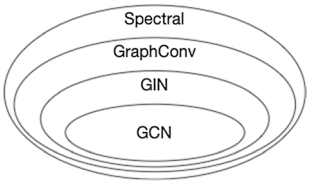

Fig. 3 Expressiveness of commonly used graph convolution operators. Each ellipse represents the set of functions that can be implemented by a single graph convolution operator

layer in a neural network. SGC (Wu et al. 2019) proposes a fast way to compute the result of several linearly stacked GCNs. Note that SGC considers just the GCN convolution, while our proposed PGC is more expressive than any number of linearly stacked graph convolutions among the ones presented in Section 3.1 of the main paper. DGCNN (Zhang et al. 2018) adopts a graph convolution very similar to GCN (Kipf and Welling 2017) (a slightly different propagation scheme for vertices' representations is defined, based on the random-walk graph Laplacian). While GCN is focused on node classification, DGCNN is suited for graph classification since it incorporates the readout.

A more straightforward approach in defining convolutions on graphs is PATCHY-SAN (PSCN) (Niepert et al. 2016). This approach is inspired by how convolutions are defined over images. It consists in selecting a fixed number of vertices from each graph and exploiting a canonical ordering on graph vertices. For each vertex, it defines a fixed-size neighborhood, exploiting the same vertex ordering. It requires the vertices of each input graph to be in a canonical ordering, which is as complex as the graph isomorphism problem (no polynomial-time algorithm is known).

Another interesting proposal for the convolution over the node neighborhood is GraphSage (Hamilton et al. 2017), which proposes to perform an aggregation over the neighborhoods by using sum, mean or max-pooling operators, and then perform a linear projection in order to update the node representation. In addition to that, the proposed approach exploits a particular neighbors sampling scheme. GIN (Xu et al. 2019) is an extension of GraphSage that avoids the limitation introduced by using sum, mean or max-pooling by using a more expressive aggregation function on multi-sets. DiffPool (Ying et al. 2018) is an end-to-end architecture that combines a differentiable graph encoder with its polling mechanism. Indeed, the method learns an adaptive pooling strategy to collapse nodes on the basis of a supervised criterion.

The Funnel GCNN (FGCNN) model (Navarin et al. 2020) aims to enhance the gradient propagation using a simple aggregation function and LeakyReLU activation functions. Hinging on the similarity of the adopted graph convolutional operator, that is the *GraphConv*, to the way feature space representations by Weisfeiler-Lehman (WL) Subtree Kernel (Shervashidze et al. 2011) are generated, it introduces a loss term for the output of each convolutional layer to guide the network to reconstruct the corresponding explicit WL features. Moreover, the number of filters used at each convolutional layer is based on a measure of the WL kernel complexity.

Appendix B: Expressiveness of commonly used graph convolutions

Thanks to the possibility to express commonly used graph convolutions as instances of PGC, and from the discussion in Section 3 of the paper, it is easy to characterize the expressiveness of commonly used graph convolutions. In Fig. 3 we represent the inclusion

relationships among the sets of functions which can be implemented by GCN, GIN, GraphConv, Spectral.

Appendix C: Proofs of theorems

Theorem 1 *Let us consider two linearly stacked PGC layers using the same transformation $\mathcal{T}$. The resulting linear Graph Convolutional network can be expressed by a single PGC layer.*

Proof With no loss of generality, let the first PGC being of degree k_1, while the second stacked PGC of degree k_2, i.e.

$$\mathbf{H}^{(1)} = [\mathbf{X}, \dots, \mathcal{T}(\mathbf{A})^{k_1}\mathbf{X}]\mathbf{W}^{(1)},$$
$$\mathbf{H}^{(2)} = [\mathbf{H^{(1)}}, \dots, \mathcal{T}(\mathbf{A})^{k_2}\mathbf{H^{(1)}}]\mathbf{W}^{(2)},$$

where

$$\mathbf{W}^{(i)} = \begin{bmatrix} \mathbf{W}_0^{(i)} \\ \mathbf{W}_1^{(i)} \\ \vdots \\ \mathbf{W}_{k_i}^{(i)} \end{bmatrix}, \ i = 1, 2.$$

By expanding $\mathbf{H}^{(1)}$ inside $\mathbf{H}^{(2)}$ equation, we get:

$$\begin{aligned} \mathbf{H}^{(2)} &= [[\mathbf{X}\mathbf{W}_0^{(1)} + \dots + \mathcal{T}(\mathbf{A})^{k_1}\mathbf{X}\mathbf{W}_{k_1}^{(1)}], \dots, \\ &\quad \mathcal{T}(\mathbf{A})^{k_2}[\mathbf{X}\mathbf{W}_0^{(1)} + \dots + \mathcal{T}(\mathbf{A})^{k_1}\mathbf{X}\mathbf{W}_{k_1}^{(1)}]]\mathbf{W}^{(2)} \\ &= [\mathbf{X}\mathbf{W}_0^{(1)}\mathbf{W}_0^{(2)} + \dots + \mathcal{T}(\mathbf{A})^{k_1}\mathbf{X}\mathbf{W}_{k_1}^{(1)}\mathbf{W}_0^{(2)} + \\ &\quad \dots + \mathcal{T}(\mathbf{A})^{k_2}\mathbf{X}\mathbf{W}_0^{(1)}\mathbf{W}_{k_2}^{(2)} + \dots + \\ &\quad \mathcal{T}(\mathbf{A})^{k_1+k_2}\mathbf{X}\mathbf{W}_{k_1}^{(1)}\mathbf{W}_{k_2}^{(2)}]. \end{aligned} \tag{18}$$

In this case, by defining $D_1 = \{0, .., k_1\}$, $D_2 = \{0, .., k_2\}$, and auxiliary functions $F_i()$, $i = 0, .., k_1 + k_2$, defined as

$$F_i(\mathbf{W}^{(1)}, \mathbf{W}^{(2)}) = \sum_{\substack{(z_1, z_2) \in D_1 \times D_2 \\ s.t.\ z_1 + z_2 = i}} \mathbf{W}_{z_1}^{(1)}\mathbf{W}_{z_2}^{(2)},$$

matrix $\mathbf{W}$ can be written as

$$\mathbf{W} = \begin{bmatrix} F_0(\mathbf{W}^{(1)}, \mathbf{W}^{(2)}) \\ F_1(\mathbf{W}^{(1)}, \mathbf{W}^{(2)}) \\ \dots \\ F_{k_1+k_2}(\mathbf{W}^{(1)}, \mathbf{W}^{(2)}) \end{bmatrix},$$

and consequently the hidden representation becomes

$$\mathbf{H}^{(2)} = [\mathbf{X}, \mathcal{T}(\mathbf{A})\mathbf{X}, \dots, \mathcal{T}(\mathbf{A})^{k_1}\mathbf{X}, \dots, \mathcal{T}(\mathbf{A})^{k_1+k_2}\mathbf{X}]\mathbf{W},$$

which is the output of a PGC with $k = k_1 + k_2$. □

Theorem 2 *A PGC layer with $k = 2$ is more general than two stacked PGC layers with $k = 1$ with the same transformation $\mathcal{T}$ and the same number of hidden units m.*

Proof Since all PGCs use the same transformation $\mathcal{T}$, we can focus on the weights only. We prove our theorem providing a counterexample, i.e. we fix $m = 1$, $s = 1$ (the input dimension) and show an instantiation of the weight matrix $\mathbf{W} = [\mathbf{W}_0^\top, \mathbf{W}_1^\top, \mathbf{W}_2^\top]^\top$ of a PGC layer with $k = 2$ that cannot be expressed by the composition of two PGC layers with $k = 1$ (equivalent to *GraphConv*). Let us consider the simplest case in which $\mathbf{W}_0, \mathbf{W}_1, \mathbf{W}_2 \in \mathbb{R}$, i.e. they are 1×1 matrices. Let us now consider the case where $\mathbf{W}_0 = 5, \mathbf{W}_1 = 7, \mathbf{W}_2 = 3$. Let us also, for the sake of clarity, rename the 1×1 matrices of the two PGCs with $k = 1$ as: $\mathbf{W}_0^{(1)} \leftarrow a, \mathbf{W}_0^{(2)} \leftarrow b, \mathbf{W}_1^{(1)} \leftarrow c, \mathbf{W}_1^{(2)} \leftarrow d$. We get the following system of equations:

$$\begin{cases} ab = 5 \\ ad + cb = 7 \\ cd = 3 \end{cases} \begin{cases} a = 5/b \\ d = 3/c \\ 15/cb + cb = 7 \end{cases} \begin{cases} a = 5/b \\ d = 3/c \\ (cb)^2 - 7cb + 15 = 0 \end{cases} \tag{22}$$

where we assume b and c are different from zero (it is easy to see that either $b = 0$ or $c = 0$ would not lead to any solution). If we compute the Δ of the third equation (solving for cb), we get $\Delta = \sqrt{49 - 60} = i\sqrt{11}$, i.e. a complex number. Thus there is no real value for cb that satisfies our system of equations. We thus conclude that there are no values that we can assign to the parameters of the PGCs with $k = 1$ that would lead to the considered PGC weight matrix. □

Moreover, Theorem 2 implies the following corollary.

Corollary 1 *A linear stack of q $\mathcal{T}$-compatible PGC layers with $k = 1$ is less expressive than a single $\mathcal{T}$-PGC layer with $k = q$.*

Proof Since all PGCs use the same transformation $\mathcal{T}$, we can focus on the weights only. We prove the corollary by induction. The base case is provided by Theorem 2. Let us now prove the inductive case. Let us assume that a stack of i PGC layers with $k = 1$ (with parameters set $\boldsymbol{\theta}_1^{(i)} = \{\mathbf{W}^{(j)} \in \mathbb{R}^{2s \times m}, j = 0, \dots, i\}$) is less expressive than a single PGC layer with $k = i$ (with parameters set $\boldsymbol{\theta}_2^{(i)} = \mathbf{W} \in \mathbb{R}^{(i+1)s \times m}$), and let us prove the same result for $i + 1$. We can consider the set of functions that can be implemented by the stack of $i + 1$ PGC layers with $k = 1$ as the composition of two functions coming from two differrent sets: the first one $PGC_{k=1}^{(i)} = \{f \mid \exists\, \hat{\boldsymbol{\theta}}_1^{(i)}\ s.t. f \equiv PGC_{k=1}^{(i)}(\hat{\boldsymbol{\theta}}_1^{(i)})\}$, is the set of functions that can be computed stacking i PGC layers with $k = 1$, and the second one $PGC_{k=1}^{(1)} = \{f \mid \exists\, \hat{\boldsymbol{\theta}}_1^{(1)}\ s.t. f \equiv PGC_{k=1}^{(1)}(\hat{\boldsymbol{\theta}}_1^{(1)})\}$, is the set of functions computed by a single PGC layer with $k = 1$. We can then characterize the set of functions $PGC_{k=1}^{(i+1)}$ as:

$$PGC_{k=1}^{(i+1)} = \{f \circ g \mid f \in PGC_{k=1}^{(i)}, g \in PGC_{k=1}^{(i)}\}.$$

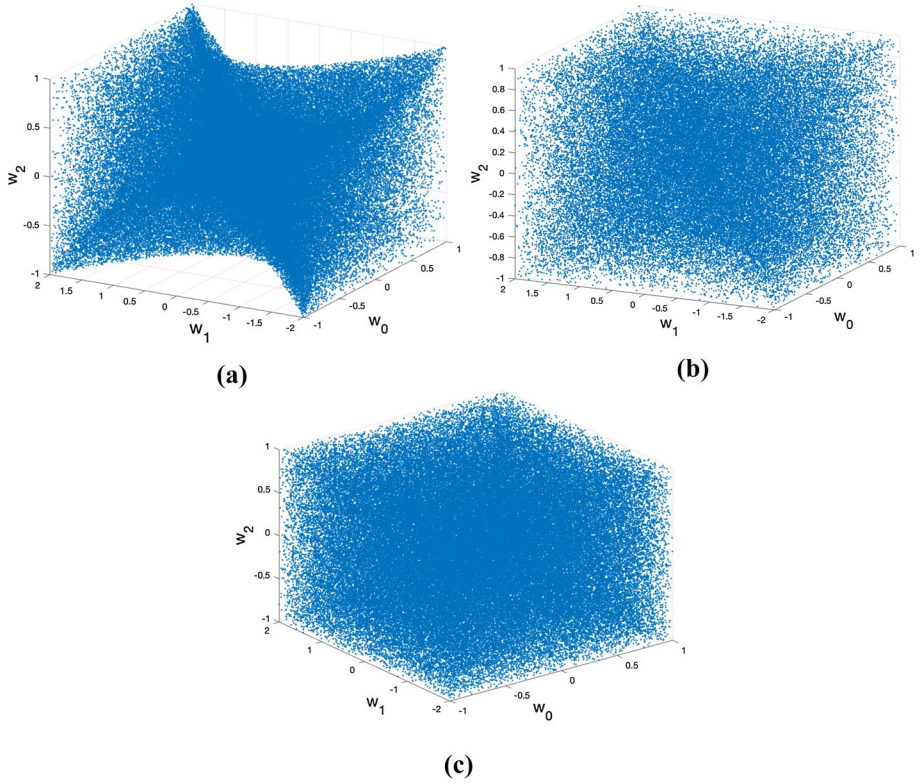

Fig. 4 Hypothesis that can be represented by two stacked *Graphconv* layers (mapped to the three PGC dimensions as per Eq. (16)) with **a** a single hidden neuron (4 trainable parameters), **b** two hidden neurons (8 parameters), and **c** by a single PGC neuron with $k = 2$ (3 parameters)

From Theorem 1, we know that $PGC^{(1)}_{k=i+1} \supseteq \{f \circ g \,|\, f \in PGC^{(1)}_{k=i}, g \in PGC^{(1)}_{k=1}\}$. Since we know that $PGC^{(1)}_{k=i} \supset PGC^{(i)}_{k=1}$ (it is more general), we conclude that $PGC^{(1)}_{k=i+1} \supset PGC^{(i+1)}_{k=1}$. □

Appendix D: Parameter inefficiency of layer stacking

In "Appendix C", we provided a proof of the limitations in the expressiveness of stacking two PGC layers with $k = 1$ compared to using a single PGC layer with $k = 2$. Here we provide additional evidence showing that the number of hypothesis that cannot be expressed by the stacked solution is not negligible. In particular, there are some regions in the hypothesis space of a single PGC neuron with $k = 2$ that cannot be covered by stacking two PGC neurons with $k = 1$.

In order to empirically show this, we conducted numerical simulations. We randomly sampled the values of the parameters of two stacked single-neurons PGC layers with $k = 1$ (equivalent to stacked *Graphconv* layers). Exploiting Eq. (16), we mapped

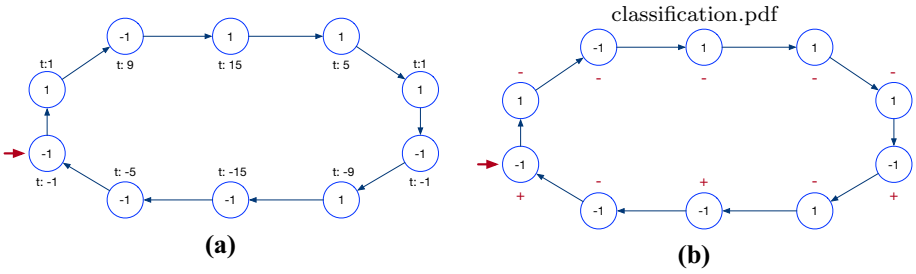

Fig. 5 Example of a regression (**a**) and a classification task (**b**) on the same graph. Node attributes are reported inside the nodes (i.e. x_u), while the desired target t is depicted close to the corresponding node for regression in (**a**) and in red as $\{+, -\}$ for binary classification in (**b**)

the solutions to the 3-dimensional space of PGC as per Eq. (17). In Fig. 4a we plot a point for each sampled hypothesis. It is clear that there are uncovered areas of the hypothesis space. In particular, it is not possible to obtain solutions where w_0 and w_1 are strongly negative (e.g. both taking the value -1) and at the same time having w_1 close to zero. As proven in Theorem 2, there are no values for the parameters of the stacked *Graphconv* layers that can represent this solution.

In order to express such solutions with stacking, we have to increase the number of neurons in the hidden representation. In Fig. 4b, the hypotheses obtained using two Graphconv neurons in the hidden representation are shown. Finally, Fig. 4c shows the hypotheses obtainable by a PGC layer with $k = 2$, i.e. the whole hypothesis space is covered. Notice that the difference in the distribution of points between Fig. 4b, c does not indicate a difference in the representable hypotheses, but it is due to the mechanism in which the parameters are combined in the stacking.

Appendix E: Example of task where stacking is not a good idea

We provide in Fig. 5a a very simple example of a task of regression (that can be interpreted as the optimal representation that a Graph Convolution Layer, or a stack of them, should learn). Since we use this example to show the difference in expressiveness between a single PGCN layer with k = 2 and a stack of 2 Graphconv layers, we will consider all the nodes as training set. The learning of a single PGCN neuron with k = 2 (thus with 3 parameters to learn) can easily achieve zero loss on the training data. On the contrary, a stack of 2 Graphconv neurons, with a total of four parameters, was not able to successfully learn the task, obtaining a training loss (Mean Squared Error) of 0.2264 after 10, 000 epochs of the Adam optimizer. Starting from the node indicated by a red arrow in Fig. 5a and moving clockwise, the predictions of Graphconv are: $[-0.3376, 0.3376, 9.4411, 14.7558, 4.9771, 0.3376, -0.3376, -9.4411, -14.7558, -4.9771]$, compared to a perfect reconstruction of the PGC layer. In fact, the PGC layer with $k = 2$ is able to learn the optimal weight vector $[3, 7, 5]^\top$, while the two stacked Graphconv architecture (if the weights are mapped back as per Eq. (16)) converges at $[2.6574, 7.2091, 4.8894]^\top$. To learn correctly such a function with the stacking mechanism, we have to exploit two *Graphconv* neurons in the first hidden layer, obtaining a

Table 7 Inputs for the first ($[x_u, (\mathbf{Ax})_u]$) and second ($[h_u, (\mathbf{Ah})_u]$) PGC convolution with $k = 1$ for the classification task shown in Fig. 5b. Row a refers to the node pointed by the red arrow in the figure. Subsequent rows refers to nodes in the clockwise order over the graph. Here we define $h_u = w_0 + w_1 = v_1$ for input $[+1, +1]$ and $h_u = w_0 - w_1 = v_2$ for input $[+1, -1]$. The last column reports the target

Id	x_u	$(\mathrm{Ax})_u$	h_u	$(\mathrm{Ah})_u$	Target
a	-1	$+1$	$-v_2$	$+v_2$	$+$
b	$+1$	-1	$+v_2$	$-v_2$	$-$
c	-1	$+1$	$-v_2$	$+v_1$	$-$
d	$+1$	$+1$	$+v_1$	$+v_1$	$-$
e	$+1$	$+1$	$+v_1$	$+v_2$	$-$
f	$+1$	-1	$+v_2$	$-v_2$	$-$
g	-1	$+1$	$-v_2$	$+v_2$	$+$
h	$+1$	-1	$+v_2$	$-v_1$	$-$
i	-1	-1	$-v_1$	$-v_1$	$+$
l	-1	-1	$-v_1$	$-v_2$	$-$

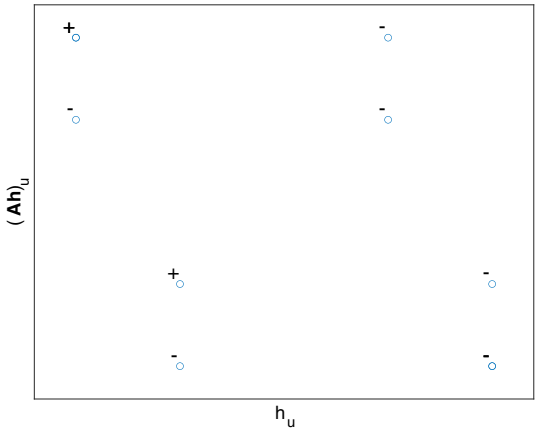

Fig. 6 Example of space in which the second *Graphconv* neuron should perform a linear classification. The example refers to the case when $0 < v_1 < v_2$

total of eight parameters to learn, i.e. more than double the number required by the PGC neuron with $k = 2$ (i.e. 3).

A similar example can be provided for a classification problem, and is reported in Fig. 5b. Node attributes x_v are reported inside the nodes. The function to learn is defined (for each node u) as: $t(u) = 1 \iff x_u = (\mathbf{A}^2\mathbf{x})_u = -1$, 0 *otherwise*, that is the function evaluates to one for a node if and only if both the labels of the node and of its neighbor at distance two (always exactly one in our example) are -1. Also in this case, a single PGC neuron with $k = 2$ can successfully learn such a function with zero training error, while two stacked single-neuron PGC layers with $k = 1$ achieve 0.8 training accuracy at most.

To understand why this is the case, we show that the input to the second single-neuron PGC layer with $k = 1$ is non-linearly separable, so it cannot perform the correct classification. In fact, if we consider a single PGC neuron with $k = 1$ defined by Eq. (9), we notice that, given that in the graph in Fig. 5b there are only two values for the node attributes (i.e., -1 and 1), the output of the first layer can take only four different values, let's say $v_1, -v_1, v_2, -v_2$. In Table 7, we report both the input to the first

neuron (i.e., $[x_u, (\mathbf{Ax})_u]$), and the input to the second neuron (i.e., $[h_u, (\mathbf{Ah})_u]$), provided that $h_u = w_0 + w_1 = v_1$ for input $[+1, +1]$ and $h_u = w_0 - w_1 = v_2$ for input $[+1, -1]$. The target for each node is reported in the last column.

We can visualize the distribution of the points and their label. In Fig. 6 we report the case in which $0 < v_1 < v_2$. It is possible to see from the plot that there is no hyperplane that can correctly classify all the training examples. The other cases show a similar situation, thus are not reported.

Appendix F: PGCN implementation details

We implemented the PGCN in PyTorch-Geometric (Fey and Lenssen 2019). To reduce the *covariate shift* during training and to attenuate overfitting, we applied batch normalization and dropout on the output of each $\mathbf{y}_j$ layer. We used the *Negative Log Likelihood* loss, the *Adam* optimizer (Kingma and Ba 2014), and the identity function for $\mathcal{T}$. For more details please check the publicly available code.[3]

For our experiments, we adopted two types of machines, respectively equipped with:

- 2 x Intel(R) Xeon(R) CPU E5-2630L v3, 192GB of RAM and a Nvidia Tesla V100;
- 2 x Intel(R) Xeon(R) CPU E5-2650 v3, 160 GB of RAM and Nvidia T4.

Appendix G: Datasets

We empirically validated the proposed PGC-GNN on five commonly adopted graph classification benchmarks modeling bioinformatics problems: PTC (Helma et al. 2001), NCI1 (Wale et al. 2008), PROTEINS, (Borgwardt et al. 2005), D&D (Dobson and Doig 2003) and ENZYMES (Borgwardt et al. 2005). The first two of them contains chemical compounds represented by their molecular graph, where each node is labeled with an atom type, and the edges represent bonds between them. PTC contains chemical compounds and the task is to predict their carcinogenicity for male rats. In NCI1 the graphs represent anti-cancer screens for cell lung cancer. The last three datasets, PROTEINS, D&D and ENZYMES, contain graphs that represent proteins. Each node corresponds to an amino acid and an edge connects two of them if they are less then 6Å (Angstrom) apart. In particular ENZYMES, differently than the other considered datasets (that model binary classification problems) allows testing the model on multi-class classification over 6 classes. We additionally considered three large social graph datasets: COLLAB, IMDB-B, IMDB-M (Yanardag and Vishwanathan 2015). In COLLAB each graph represents a collaboration network of a corresponding researcher with other researchers from three fields of physics. The task consists in predicting the physics field the researcher belongs to. IMDB-B and IMDB-M are composed of graphs derived from actor/actress who played in different movies on IMDB, together with the movie genre information. Each graph has a target that represents the movie genre. IMDB-B models a binary classification task, while IMDB-M contains graphs that belong to three different classes. Differently from the bioinformatics

[3] https://github.com/lpasa/PGCN

Table 8 Datasets statistics

Dataset	#Graphs	#Node	#Edge	Avg #Nodes/graph	Avg.#Edges/graph
PTC	344	4915	10108	14.29	14.69
NCI1	4110	122747	265506	29.87	32.30
PROTEINS	1113	43471	162088	39.06	72.82
D&D	1178	334925	1686092	284.32	715.66
ENZYMES	600	19580	74564	32.63	124.27
COLLAB	5000	372474	24572158	74.50	4914.43
IMDB-B	1000	19773	193062	19.773	193.06
IMDB-M	600	19502	197806	13.00	131.87

Table 9 Average ratio of the number of null entries over the total number of entries in the input components up to $k = 5$ without (top row) and with (bottom) the $PGC_{k=1}$ layer for the used datasets

Input	PTC	NCI1	PROTEINS	D&D	ENZYMES
$\mathbf{X}$	0.94	0.97	0.67	0.99	0.26
$PGC_{k=1}(\mathbf{X})$	0	0	0	0	0
$\mathbf{AX}$	0.93	0.96	0.45	0.95	0.13
$\mathbf{A}\, PGC_{k=1}(\mathbf{X})$	0	$3.67 \cdot 10^{-3}$	$9.28 \cdot 10^{-5}$	0	$1.87 \cdot 10^{-3}$
$\mathbf{A}^2\mathbf{X}$	0.90	0.95	0.38	0.89	0.11
$\mathbf{A}^2\, PGC_{k=1}(\mathbf{X})$	0	$3.67 \cdot 10^{-3}$	$9.28 \cdot 10^{-5}$	0	$1.87 \cdot 10^{-3}$
$\mathbf{A}^3\mathbf{X}$	0.89	0.95	0.36	0.86	0.10
$\mathbf{A}^3\, PGC_{k=1}(\mathbf{X})$	0	$3.67 \cdot 10^{-3}$	$9.28 \cdot 10^{-5}$	$2.39 \cdot 10^{-8}$	$1.87 \cdot 10^{-3}$
$\mathbf{A}^4\mathbf{X}$	0.88	0.94	0.35	0.87	0.10
$\mathbf{A}^4\, PGC_{k=1}(\mathbf{X})$	0	$3.67 \cdot 10^{-3}$	$9.28 \cdot 10^{-5}$	0	$1.87 \cdot 10^{-3}$
$\mathbf{A}^5\mathbf{X}$	0.8	0.94	0.35	0.81	0.10
$\mathbf{A}^5\, PGC_{k=1}(\mathbf{X})$	0	$3.67 \cdot 10^{-3}$	$9.28 \cdot 10^{-5}$	$4.71 \cdot 10^{-8}$	$1.87 \cdot 10^{-3}$

The value 0 corresponds to a dense matrix, while the value 1 to a null matrix

datasets, the nodes contained in the social datasets do not have any associated label. Relevant statistics about the datasets are reported in Table 8.

Appendix H: PGCN computation complexity example

Consider a dataset with n_G graphs, and the 2-layers *GraphConv* defined with a message passing formulation in Eqs. (14) and (15) (assuming $m_1 = m_2 = m$). Each *GraphConv* layer requires 3 matrix multiplications. The $\mathbf{AX}$ term in the first layer can be pre-computed since it remains the same over all training. Thus a 2-layer GCN performs $5 \cdot n_G$ matrix multiplications in the forward pass for each epoch (generally the size of $\mathbf{A}$ is different for each graph, but for the sake of discussion we can assume their dimension is comparable). Assuming 100 epochs for training, the total number of such multiplications is then $5 \cdot 100 \cdot n_G + 1$. If we now consider the PGC formulation with $k = 2$ in Eq. (17) (that we

recall is more expressive than two stacked *GraphConv* layers, as shown in Sect. 4.1), the number of matrix multiplications required for each graph is 6. However, the terms $\mathbf{AX}$ and $\mathbf{A}^2\mathbf{X}$ remain the same, for each graph, during all the training. They can thus be pre-computed and stored in memory. With this implementation, Eq. (17) would require just three matrix multiplications, for a total number of matrix multiplications for 100 training epochs of $3 \cdot 100 \cdot n_G + 3$. While this does not modify the asymptotic complexity of PGC compared to message passing, it significantly improves the training times.

Appendix I: Initial node embeddings

Some datasets that we used in the experiments, encode node labels (i.e., $\mathbf{X}$) by using a one-hot encoding. That makes the nodes representations very sparse. In preliminary experiments, we observed that such sparse representations negatively influence learning. In Table 9, we show how the use of a sparse node representation as input leads to have sparse input matrices $\mathbf{X}, \mathbf{AX}, \ldots, \mathbf{A}^k\mathbf{X}$. Specifically, in order to estimate the difference in terms of the sparsity degree with or without an initial PGC layer with $k = 1$, we computed the average ratio between the number of null entries (we round all the embedding values to the 4th decimal digit) and the total number of entries of the input matrices on the whole dataset for all the used bioinformatics datasets. We evaluated the sparsity of each PCG-layer block, considering the values of k in the interval $[0, \ldots, 5]$. It is interesting to notice that in all datasets the use of the initial PGC leads to a sparsity ratio near 0 (therefore the subsequent PGC-layer has in input dense embeddings). That is very useful, in particular for datasets like NCI1, PTC, and D&D, where the percentage of zeros in the labels representation is near 90%.

Appendix J: Readout in literature

For what concerns the readout stage we decided to adopted a simple and common architecture in order to obtain comparable results with the literature and to highlight the benefit of using the proposed Polynomial Graph Convolution. Indeed, we use a simple MLP, experimenting with two alternatives: (i) a single layer readout and (ii) an MLP composed of three fully connected readout layers. For what concerns the aggregation step, we use the concatenation of three simple element-wise operations (sum, mean, and max), that is the same aggregation used by FGCNN. Note that many of the of the architectures considered in the comparison use a more complex readout. For instance, the DIFFPOOL uses a pretty complex node pooling method, while DGCNN uses the SortPooling layer followed by a 1D-convolutional layer followed by several dense layers (MLP).

Table 10 Sets of hyper-parameters values used for model selection via grid search

Dataset/k	m	Learning rate	Weight decay	Drop out	Batch size	k	Readout(#layers [dims])
PTC	15, 30, 60	$10^{-3}, 5 \cdot 10^{-4}, 10^{-4}$	$5 \cdot 10^{-3}, 5 \cdot 10^{-4}$	0.4, 0.6	16, 32	2, 3, 4, 5	1 [$m/2$], 2 [$m * 2, m$]
NCI1	50, 100	$10^{-3}, 5 \cdot 10^{-4}$	$5 \cdot 10^{-3}, 5 \cdot 10^{-4}$	0.3, 0.5	16, 32	2, 3, 4, 5	1 [$m/2$], 2 [$m * 2, m$]
PROTEINS	25, 50	$10^{-3}, 5 \cdot 10^{-4}, 10^{-4}$	$5 \cdot 10^{-3}, 5 \cdot 10^{-4}$	0.3, 0.5	16, 32	2, 3, 4, 5	1 [$m/2$], 2 [$m * 2, m$]
D&D	50, 75	$5 \cdot 10^{-4}, 5 \cdot 10^{-5}$	$5 \cdot 10^{-3}, 5 \cdot 10^{-4}$	0.3, 0.5	16, 32	2, 3, 4, 5	1 [$m/2$], 2 [$m * 2, m$]
ENZYMES	50, 100	$10^{-3}, 10^{-4}$	$5 \cdot 10^{-3}, 5 \cdot 10^{-4}$	0.3, 0.5	16, 32	2, 3, 4, 5	1 [$m/2$], 2 [$m * 2, m$]
COLLAB	7, 15, 30	$10^{-3}, 5 \cdot 10^{-4}$	$5 \cdot 10^{-3}, 5 \cdot 10^{-4}$	0, 0.5	16, 32	2, 4, 6, 8	1 [$m/2$], 2 [$m * 2, m$]
IMDB-B	50, 75	$10^{-4}, 10^{-5}$	$5 \cdot 10^{-4}, 5 \cdot 10^{-5}$	0, 0.5	16, 32	2, 4, 6, 8	1 [$m/2$], 2 [$m * 2, m$]
IMDB-M	50, 75, 100	$10^{-4}, 5 \cdot 10^{-5}$	$5 \cdot 10^{-3}, 5 \cdot 10^{-4}$	0, 0.5	16, 32	2, 4, 6, 8	1 [$m/2$], 2 [$m * 2, m$]

Appendix K: Hyper-parameters selection

Due to the high computational time required to perform an extensive grid search, we decided to limit the number of values taken into account for each hyper-parameter, by performing preliminary tests to identify useful ranges of values.

The hyper-parameters of the model (number of hidden units, learning rate, weight decay, k) were selected by using a limited grid search, where the explored sets of values do change based on the considered dataset. Due to the high time requirements of performing an extensive grid search, we decided to limit the number of values taken into account for each hyper-parameter, by performing some preliminary tests.Preliminary tests showed that for the social network datasets, it is more convenient to use the Laplacian $\mathbf{L}$ as $\mathcal{T}(\mathbf{A})$. This behavior could be due to lack of label associated to nodes. In Table 10, we report the sets of hyper-parameters values used for model selection via grid search. As evaluation measure, we used the average accuracy computed over the 10-fold cross-validation on the validation sets, and we used the same set of selected hyper-parameters for each fold. For what concerns the selection of the epoch, it was performed for each fold independently based on the accuracy value on the validation set.

Acknowledgements The authors acknowledge the HPC resources of the Department of Mathematics, University of Padua, made available for conducting the research reported in this paper.

Author contributions LP conceptualization, methodology, software, experimental evaluation, writing; NN conceptualization, methodology, writing; AS conceptualization, methodology, writing, supervision.

Funding This work was supported by the Department of Mathematics, University of Padua, through the SID/BIRD 2020 Project "Deep Graph Memory Networks."

Availability of data and material and code availability The datsets and the code that support the findings of this study are openly available in *PyTorch-Geometric* library, and in the following repository: https://github.com/lpasa/PGCN. https://github.com/lpasa/MultiScale_GCNs

Declarations

Conflict of interests The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Consent for publication Not applicable

Ethics approval and Consent to participate Not applicable

References

Abu-El-Haija, S., Perozzi, B., Kapoor, A., Alipourfard, N., Lerman, K., Harutyunyan, H., Steeg, G.V., & Galstyan, A. (2019). Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In K. Chaudhuri, R. Salakhutdinov (Eds.) *Proceedings of the 36th international conference on machine learning, ICML 2019, 9–15 June 2019, Long Beach, California, USA, PMLR, Proceedings of machine learning research* (Vol. 97, pp 21–29). http://proceedings.mlr.press/v97/abu-el-haija19a.html.

Atwood, J., & Towsley, D. (2016). Diffusion-convolutional neural networks. In *Advances in neural information processing systems* (pp 1993–2001).

Borgwardt, K.M., Ong, C.S., Schönauer, S., Vishwanathan, S., Smola, A.J., & Kriegel, H.P. (2005). Protein function prediction via graph kernels. *Bioinformatics* 21(suppl_1):i47–i56.
Chen, T., Bian, S., & Sun, Y. (2019). Are powerful graph neural nets necessary? A dissection on graph classification. arXiv preprint arXiv:190504579.
Defferrard, M., Bresson, X., & Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS* (pp. 3844–3852).
Dobson, P. D., & Doig, A. J. (2003). Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology, 330*(4), 771–783.
Errica, F., Podda, M., Bacciu, D., & Micheli, A. (2020). A fair comparison of graph neural networks for graph classification. In *International conference on learning representations*
Fey, M., & Lenssen, J.E. (2019). Fast graph representation learning with PyTorch Geometric. In *ICLR workshop on representation learning on graphs and manifolds*
Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., & Dahl, G.E. (2017). Neural message passing for quantum chemistry. In *Proceedings of the 34th international conference on machine learning* (pp 1263–1272), arXiv:1704.01212
Hamilton, W., Ying, Z., & Leskovec, J. (2017). Inductive representation learning on large graphs. In *NIPS* (pp 1024–1034).
Helma, C., King, R. D., Kramer, S., & Srinivasan, A. (2001). The predictive toxicology challenge 2000–2001. *Bioinformatics, 17*(1), 107–108.
Japkowicz, N., & Shah, M. (2011). Evaluating Learning Algorithms. *Cambridge University Press*https://doi.org/10.1017/CBO9780511921803, *http://ebooks.cambridge.org/ref/id/CBO9780511921803, citation Key: Japkowicz 2011 ISSN: 1098-6596*
Kingma, D.P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:14126980.
Kipf, T.N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In ICLR (pp. 1–14). https://doi.org/10.1051/0004-6361/201527329. arXiv:1609.02907
Klicpera, J., Bojchevski, A., & Günnemann, S. (2019). Predict then propagate: Graph neural networks meet personalized pagerank. In *7th international conference on learning representations, ICLR 2019*, New Orleans, LA, USA, May 6-9, 2019, OpenReview.net.
Li, Q., Han, Z., & Wu, X.M. (2018). Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI conference on artificial intelligence*.
Liao, R., Zhao, Z., Urtasun, R., & Zemel, R.S. (2019). Lanczosnet: Multi-scale deep graph convolutional networks. In *7th international conference on learning representations, ICLR 2019*.
Liu, M., Gao, H., & Ji, S. (2020). Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining* (pp. 338–348)
Luan, S., Zhao, M., Chang, X.W., & Precup, D. (2019). Break the ceiling: Stronger multi-scale deep graph convolutional networks. In *Advances in neural information processing systems* (pp. 10945–10955).
Micheli, A. (2009). Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks, 20*(3), 498–511.
Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., & Grohe, M. (2019). Weisfeiler and Leman Go Neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence, 33*, 4602–4609. https://doi.org/10.1609/aaai.v33i01.33014602arXiv:1810.02244.
Navarin, N., Van Tran, D., & Sperduti, A. (2020). Learning kernel-based embeddings in graph neural networks. In: 24th European conference on artificial intelligence - ECAI 2020.
Niepert, M., Ahmed, M., & Kutzkov, K. (2016). Learning convolutional neural networks for graphs. In: *ICML*, pp 2014–2023.
Rossi, E., Frasca, F., Chamberlain, B., Eynard, D., Bronstein, M., & Monti, F. (2020). Sign: Scalable inception graph neural networks. arXiv preprint arXiv:200411198.
Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2008). The graph neural network model. *IEEE Transactions on Neural Networks, 20*(1), 61–80.
Shervashidze, N., Schweitzer, P., Van Leeuwen, E. J., Mehlhorn, K., & Borgwardt, K. M. (2011). Weisfeiler–Lehman graph kernels. *Journal of Machine Learning Research, 12*(77), 2539–2561.
Sperduti, A., & Starita, A. (1997). Supervised neural networks for the classification of structures. *IEEE Trans Neural Networks, 8*(3), 714–735. https://doi.org/10.1109/72.572108
Tran, D.V., Navarin, N., & Sperduti, A. (2018). On filter size in graph convolutional networks. In *2018 IEEe symposium series on computational intelligence (SSCI), IEEE* (pp. 1534–1541).
Wale, N., Watson, I. A., & Karypis, G. (2008). Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems, 14*(3), 347–375.

Wu, F., Zhang, T., de Souza, A. H., Fifty, C., Yu, T., & Weinberger, K. Q. (2019). Simplifying graph convolutional networks. *ICML, 1902,* 07153.

Xu, K., Li, C., Tian, Y., Sonobe, T., Ki, Kawarabayashi, & Jegelka, S. (2018). Representation learning on graphs with jumping knowledge networks. *PMLR, Stockholmsmässan, Stockholm Sweden, Proceedings of Machine Learning Research, 80*, 5453–5462.

Xu, K., Hu, W., Leskovec, J., & Jegelka, S. (2019). How powerful are graph neural networks? In *International conference on learning representations*.

Yanardag, P., & Vishwanathan, S. (2015). Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 1365–1374).

Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., & Leskovec, J. (2018). Hierarchical graph representation learning with differentiable pooling. In *Advances in neural information processing systems* (pp. 4800–4810).

Zhang, M., Cui, Z., Neumann, M., & Chen, Y. (2018). An end-to-end deep learning architecture for graph classification. In *Thirty-second AAAI conference on artificial intelligence*.