Empowering Simple Graph Convolutional Networks

Luca Pasa, Nicolò Navarin, *Member, IEEE*, Wolfgang Erb, and Alessandro Sperduti, *Senior Member, IEEE*

***Abstract*— Many neural networks for graphs are based on the graph convolution (GC) operator, proposed more than a decade ago. Since then, many alternative definitions have been proposed, which tend to add complexity (and nonlinearity) to the model. Recently, however, a simplified GC operator, dubbed simple graph convolution (SGC), which aims to remove nonlinearities was proposed. Motivated by the good results reached by this simpler model, in this article we propose, analyze, and compare simple graph convolution operators of increasing complexity that rely on linear transformations or controlled nonlinearities, and that can be implemented in single-layer graph convolutional networks (GCNs). Their computational expressiveness is characterized as well. We show that the predictive performance of the proposed GC operators is competitive with the ones of other widely adopted models on the considered node classification benchmark datasets.**

***Index Terms*— Deep learning, graph convolution (GC), graph neural network (GNN), machine learning on graphs, structured data.**

I. Introduction

In THE past few years, there has been an increasing interest in the machine learning models able to deal with graph-structured data, including kernel methods [1] and neural networks [2]. The idea of graph neural networks (GNNs) is to define a neural architecture that follows the topology of the graph. Then a transformation is performed from the neurons corresponding to a vertex and its neighborhood to a hidden representation, which is associated with the same vertex in another layer of the network. A new transformation is then performed for each hidden layer of the GNN. Each of these transformations depends on some parameters, which may be shared among all the vertices, obtaining graph convolutional networks (GCNs).

All these models share the intuition that nonlinearities are essential to obtain models with high accuracy. Recently, this idea has been questioned by the proposal of simple graph convolution (SGC) [3], which remove the nonlinearities from a popular GCN model, showing that it did not significantly impact on the resulting predictive performance.

Manuscript received 21 August 2021; revised 9 May 2022 and 7 November 2022; accepted 15 December 2022. This research was supported by the Department of Mathematics, University of Padua with the SID/BIRD 2020 project "Deep Graph Memory Networks" and with the provision of the necessary High Performance Computing (HPC) resources. *(Corresponding author: Luca Pasa.)*

Luca Pasa, Nicolò Navarin, and Wolfgang Erb are with the Department of Mathematics "Tullio Levi-Civita," University of Padua, 35121 Padua, Italy (e-mail: lpasa@math.unipd.it).

Alessandro Sperduti is with the Department of Mathematics "Tullio Levi-Civita," University of Padua, 35121 Padua, Italy, and also with the DISI, University of Trento, 38123 Trento, Italy.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNNLS.2022.3232291.

Digital Object Identifier 10.1109/TNNLS.2022.3232291

The SGC convolution is defined to have the minimum number of parameters possible, which is the input size multiplied by the output size [4]. While this simple structure offers a considerable speed-up of the training process, it might limit the model in terms of expressiveness and the class of functions that can be learned. In this article, we study the role of such a simple convolutional filter and investigate how the adjunction of additional parameters and nonlinearities in the filter function influences the explored hypothesis space and the efficiency of the method. Pursuing this aim, we introduce simple convolutional models similar to the SGC that are able to represent overlapping and larger classes of functions. Maintaining the SGC approach of not being based on message-passing, but defining a single layer that considers a variable receptive field, we explore a landscape of related graph convolutional filters with increasing complexity and expressiveness to better understand, empirically and theoretically, how they behave in terms of expressiveness, efficiency, and efficacy when applied to tasks of increasing complexity.

As a first alternative to SGC, we consider a convolutional model based on one of the simplest and most widespread linear recursive filters, i.e., the exponential filter. Compared with SGC, the resulting exponential graph convolution (ExpGC) exploits one additional filter parameter that can be learned and interpreted as a diffusion rate on the graph. ExpGC shows a competitive and in some cases a better predictive performance than SGC. In general, the classes of functions represented by ExpGC and SGC are, up to particular cases, disjunct.

As a still simple but more expressive and general linear model, we then consider linear graph convolution (LGC) [5]. The LGC model incorporates $k+1$ parameters as coefficients of a polynomial convolutional filter. Neglecting a slightly differing normalization factors in the convolutional operation, the LGC model is more general than ExpGC and SGC and includes all the functions that can be represented by ExpGC and SGC.

We then study a new way of increasing the expressive power of such simple convolutions by defining a mechanism based on simple hypernetworks to dynamically adapt the convolution parameters to the input node, obtaining Hyper-ExpGC (hExpGC) and Hyper-LGC (hLGC). We provide an analysis of the expressiveness (i.e., the class of functions that can be represented by the applied convolutions) and the Rademacher complexity of our proposed simple graph convolutions, showing the hierarchical relationship among their hypothesis spaces. Having several increasingly complex hypothesis spaces to exploit (from different graph convolutions) allows to consider the specific convolution as an hyperparameter, finding the most suitable model for the task at hand. The relationships among the classes of functions that can

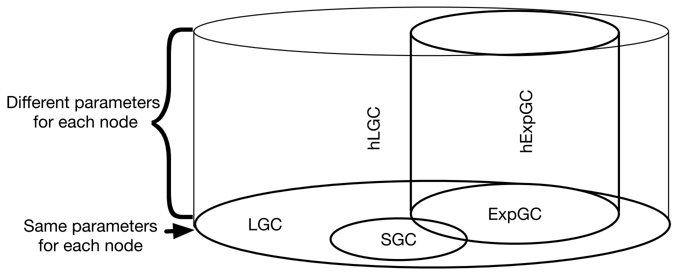

Different parameters for each node
Same parameters for each node
hLGC
hExpGC
LGC
SGC
ExpGC

Fig. 1. Relationships among the classes of functions that can be represented by different graph convolutions. The circles represent the class of functions that the various convolutional operators can model. We use cylinders to describe the ability of the hypernetwork-based architectures to have different parameters for each node.

be represented by the different convolutions are summarized in Fig. 1.

We also show that hExpGC is able to provide a hint about what is the neighbor size (i.e., how many hops from the node should be performed) most relevant to the generation of the output, thus providing a simple explainability mechanism.

From the experimental point of view, we considered six datasets in our experimental comparison and show that the proposed models achieve competitive predictive performance. Finally, we include a detailed comparison on the computational requirements of our proposed methods, showing that three of our proposals are among the fastest methods in literature. Through the experimental results, we also investigate when it is more convenient to use a simpler model to maximize the ratio between computation time and performance.

II. Background

In the following, we denote scalars with lowercase letters, e.g., x, vectors with bold lowercase letters, e.g., $\mathbf{a}$, and matrices with bold uppercase letters, e.g., $\mathbf{M}$. When referring to the elements of a matrix, we use the row and column indices as subscripts, and the same letter is used for the matrix in lowercase, i.e., m_{ij} denotes the element at the ith row, jth column of $\mathbf{M}$. Furthermore, we denote sets with uppercase letters, e.g., S.

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 $\mathbf{X} \in \mathbb{R}^{n \times c}$ is a multivariate signal on the graph nodes with the ith 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 $a_{ij} = 1 \iff (v_i, v_j) \in E$ and $a_{ij} = 0$ otherwise.

A. Graph Convolutions

The derivation of the graph convolution (GC) operator originates from graph spectral filtering [6], [7]. Let us fix a graph G. Let $\mathbf{x} : V \to \mathbb{R}$ be a signal on the nodes V of the graph G, i.e., a function that associates a real value to each node of V. Since the number of nodes in G is fixed (i.e., n) and the set V is ordered, we can naturally represent every signal as a vector $\mathbf{x} \in \mathbb{R}^n$. 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 $\mathbf{f}$. However, as we do not have an inherent description of translation on G, it is not so obvious how to define the convolution directly in the graph domain. This operation is therefore usually defined in the spectral domain of the graph, using an analogy to classical Fourier analysis in which the convolution of two signals is calculated as the pointwise product of their Fourier transforms.

For this reason, we first provide a definition of the graph Fourier transform [8]. Let $\mathbf{L}$ be the (normalized) graph Laplacian, defined as $\mathbf{L} = \mathbf{I_n} - \mathbf{D}^{-(1/2)}\mathbf{A}\mathbf{D}^{-(1/2)}$, where $\mathbf{I_n}$ is the $n \times n$ identity matrix, and $\mathbf{D}$ is the degree matrix. Since $\mathbf{L}$ is real, symmetric, and positive semi-definite, we can compute its eigendecomposition as $\mathbf{L} = \mathbf{U}\Lambda\mathbf{U}^\top$, where Λ is a diagonal matrix with the ordered eigenvalues of $\mathbf{L}$ as diagonal entries, and the orthonormal matrix $\mathbf{U}$ contains the corresponding eigenvectors $\{\mathbf{u}_0, \ldots, \mathbf{u}_{n-1}\}$ of $\mathbf{L}$ as columns. In particular, going back to our spatial signal $\mathbf{x}$, we can define its graph Fourier transform as $\hat{\mathbf{x}} = \mathbf{U}^\top\mathbf{x}$.

Using the graph Fourier transform to switch between the spatial and spectral domains, we are now ready to define the graph convolution between a filter $\mathbf{f}$ and a signal $\mathbf{x}$ as follows:

$$\mathbf{f} *_G \mathbf{x} = \mathbf{U}\big(\hat{\mathbf{f}} \odot \hat{\mathbf{x}}\big) = \mathbf{U}\big(\big(\mathbf{U}^\top\mathbf{f}\big) \odot \big(\mathbf{U}^\top\mathbf{x}\big)\big) \tag{1}$$

where $\hat{\mathbf{f}} \odot \hat{\mathbf{x}}$ denotes the componentwise Hadamard product of the two vectors $\hat{\mathbf{x}}$ and $\hat{\mathbf{f}}$. The Hadamard product $\hat{\mathbf{f}} \odot \hat{\mathbf{x}}$ can be formulated in matrix–vector notation as $\hat{\mathbf{f}} \odot \hat{\mathbf{x}} = \hat{\mathbf{F}}\hat{\mathbf{x}}$ by applying the diagonal matrix $\hat{\mathbf{F}} = \operatorname{diag}(\hat{\mathbf{f}})$. According to (1), we therefore obtain

$$\mathbf{f} *_G \mathbf{x} = \mathbf{U}\hat{\mathbf{F}}\mathbf{U}^\top\mathbf{x}. \tag{2}$$

We can design the diagonal matrix $\hat{\mathbf{F}}$ and, thus, the spectral filter $\mathbf{f}$ in various ways. The simplest way would be to define $\mathbf{f}_{\boldsymbol{\theta}}$ as a parametric filter, i.e., use $\hat{\mathbf{F}}_{\boldsymbol{\theta}} = \operatorname{diag}(\boldsymbol{\theta})$, where $\boldsymbol{\theta} = (\theta_0, \ldots, \theta_{n-1})^\top$ is a completely free vector of filter parameters that can be learned by the neural network. However, such a filter grows in size with the data, and it is not well-suited for learning. A better option pursued in this work is to use a polynomial parameterization based on powers of the spectral matrix Λ for the filter, such as

$$\hat{\mathbf{F}}_{\boldsymbol{\theta}} = \sum_{i=0}^{k} \theta_i \Lambda^i. \tag{3}$$

This filter has $k+1$ parameters $\{\theta_0, \ldots, \theta_k\}$ to learn, and it is spatially k-localized on the graph. One of the main advantages of this filter is that we can formulate it explicitly in the graph domain. Recalling the eigendecomposition $\mathbf{L} = \mathbf{U}\Lambda\mathbf{U}^\top$ of the graph Laplacian, (2) and (3) combined give

$$\mathbf{f}_{\boldsymbol{\theta}} *_G \mathbf{x} = \mathbf{U}\hat{\mathbf{F}}_{\boldsymbol{\theta}}\mathbf{U}^\top\mathbf{x} = \textstyle\sum_{i=0}^{k} \theta_i \mathbf{L}^i\mathbf{x}. \tag{4}$$

Kipf and Welling [9] propose to fix the order $k = 1$ in (3) to obtain a linear first-order filter for each graph convolutional layer in a neural network. These simple convolutions can then be stacked to improve the discriminatory power of the resulting network.

The authors additionally use a renormalization trick to limit the eigenvalues of the resulting matrix: they replace

$\mathbf{I_n} + \mathbf{D}^{-(1/2)}\mathbf{A}\mathbf{D}^{-(1/2)}$ by $\tilde{\mathbf{D}}^{-(1/2)}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-(1/2)}$, where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I_n}$ and $(\mathbf{D})_{ii} = \sum_{j=0}^{n}(\tilde{\mathbf{A}})_{ij}$.

Applying this convolution operator to a multivariate signal $\mathbf{X} \in \mathbb{R}^{n\times c}$ and using m filters, we obtain the following definition for a single graph convolutional layer: $\mathbf{H} = \tilde{\mathbf{D}}^{-(1/2)}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-(1/2)}\mathbf{X}\mathbf{\Theta}$, where $\mathbf{\Theta} \in \mathbb{R}^{c\times m}$.

To obtain a GCN, several graph convolutional layers are stacked and interleaved by a nonlinear activation function, typically a rectified linear unit (ReLU).

If $\mathbf{H}^{(0)} = \mathbf{X}$, we obtain the following recursive definition for the kth graph convolutional layer:

$$\mathbf{H}^{(k)} = \mathrm{ReLU}\Big(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(k-1)}\mathbf{\Theta}\Big). \tag{5}$$

Although GC can also be applied to other settings, we will from now on focus on the task of multiclass classification. In the last GC layer (say the lth), the ReLU activator is replaced by a softmax classifier (that is a multinomial logistic regression) to output the predictions.

B. Simple Graph Convolution

In [3], a simplification of the convolution operator in (5) is proposed, dubbed SGC. The idea is that perhaps the nonlinear operator introduced by GCNs is not essential. However, stacking multiple GC layers has an important effect on the locality of the learned filters, i.e., after k GC layers, the hidden representation of a vertex considers information coming from the vertices up to distance k, i.e., the filters on the kth layer are k-localized. Let us rewrite, for ease of notation

$$\mathbf{S} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}} \tag{6}$$

then a GC layer as defined in (5) (not considering the ReLU nonlinearity) becomes $\mathbf{H}^{(i)} = \mathbf{S}\mathbf{H}^{(i-1)}\mathbf{\Theta}$. If we stack k such layers with no nonlinearity, and we apply a *softmax* classifier at the end, the output after k hidden layers is

$$\mathbf{Y} = \mathrm{softmax}\left(\overbrace{\mathbf{S}\ldots\mathbf{S}}^{k}\,\mathbf{X}\mathbf{\Theta}^{(0)}\ldots\mathbf{\Theta}^{(k-1)}\right). \tag{7}$$

Since the SGC model is linear, we can reparameterize it as $\mathbf{\Theta} = \Theta^{(\mathbf{0})}\ldots\Theta^{(\mathbf{k-1})}$ obtaining

$$\mathbf{Y} = \mathrm{softmax}\big(\mathbf{S}^k\mathbf{X}\mathbf{\Theta}\big). \tag{8}$$

The great advantage of this model is a reduced number of parameters compared with classical graph convolution. Moreover, $\mathbf{S}^k$ can be computed only once, with a dramatic speed-up compared with GCNs.

III. Proposed Graph Convolutions

In the following, we present the main contributions of this article. We introduce several simple convolutional models with increasing number of parameters to analyze how the model complexity influences its expressiveness and learnability of function classes. We first use (an approximate) exponential graph filter to define a respective convolutional network which we refer to as ExpGC. Compared with SGC, it exploits an additional parameter for the filter function.

We then present a more general and expressive model than ExpGC and SGC, the so-called LGC [5]. The LGC, differently than SGC and ExpGC, contains all the linear combinations of monomials up to a degree k as possible filter functions. Thanks to this general definition, the LGC model can represent larger classes of functions compared with SGC and ExpGC. For these three simple models, we present upper bounds for their Rademacher complexity. Since the considered models use a single graph convolution layer, these bounds can be directly used to estimate the generalization error.

Finally, we define even more expressive convolutions, the hExpGC and the hLGC, which introduce gating-like functions depending on a number of parameters that are linear in the size of the graph. These functions are implemented via parameterized (and learnable) networks that generate the values of the convolution coefficients depending on the considered input vertex.

A. Exponential Graph Convolution

In the following, we introduce the ExpGC operator based on the coefficients of the exponential power series. It is one of the simplest linear filters on graphs and its action to a graph signal can be interpreted as a diffusion process on the graph. Our aim is to have an extremely simple graph convolution at disposition that is theoretically grounded in graph spectral filtering and that yields a predictive performance comparable or better than SGC.

We revisit the definition of parameterized filters in (3), considering an exponential filter instead of a polynomial one

$$\hat{\mathbf{F}}_\beta = e^{\beta\mathbf{\Lambda}} = \sum_{i=0}^{\infty}\frac{\beta^i}{i!}\mathbf{\Lambda}^i. \tag{9}$$

Recalling the eigendecomposition $\mathbf{L} = \mathbf{U}\Lambda\mathbf{U}^\top$ [similar to what we did in (4)], we can derive the corresponding convolution in the graph domain

$$\mathbf{f}_\beta *_G \mathbf{x} = \mathbf{U}\left(\sum_{i=0}^{\infty}\frac{\beta^i}{i!}\Lambda^i\right)\mathbf{U}^\intercal\mathbf{x} = \sum_{i=0}^{\infty}\frac{\beta^i}{i!}\mathbf{L}^i\mathbf{x} = e^{\beta\mathbf{L}}\mathbf{x}.$$

Truncating the series to a maximum number k and applying this filter to a multivariate signal and m outputs, we obtain one layer of the ExpGC as

$$\mathbf{H}^{(k)} = \sum_{i=0}^{k}\frac{\beta^i}{i!}\mathbf{L}^i\mathbf{X}\mathbf{\Theta} \tag{10}$$

where β and $\mathbf{\Theta}$ are the parameters to be learned, and k is an hyperparameter.

Note that in the limit $k \to \infty$, we get

$$\mathbf{H}^{(\infty)} = \lim_{k\to\infty}\mathbf{H}^{(k)} = e^{\beta\mathbf{L}}\mathbf{X}\mathbf{\Theta}. \tag{11}$$

This can be derived from the following approximation error of our truncated ExpGC with respect to $\mathbf{H}_\infty$:

$$\begin{aligned}\big\|\mathbf{H}^{(\infty)} - \mathbf{H}^{(k)}\big\| &= \left\|\sum_{i=k+1}^{\infty}\frac{\beta^i}{i!}\mathbf{L}^i\mathbf{X}\mathbf{\Theta}\right\| \\ &\le \frac{|\beta|^{k+1}\|\mathbf{L}\|^{k+1}}{(k+1)!}\frac{\|\mathbf{X}\mathbf{\Theta}\|}{1-|\beta|\|\mathbf{L}\|/(k+2)}\end{aligned} \tag{12}$$

where $\|\cdot\|$ denotes the spectral norm for matrices. In particular, this estimate guarantees that for a truncation number k large enough, the polynomial ExpGC model mimics a graph convolution with an exponential kernel. As the solution of the diffusion equation on the graph is determined by the exponential kernel $e^{\beta \mathbf{L}}$, the input and output layers in the ExpGC model are linked by a diffusion process on the graph nodes. Moreover, during the training process the diffusion rate β is optimally adjusted to the given training set. This yields a first improvement over the SGC model introduced above in which the propagation matrix $\mathbf{S}^k$ was a priori fixed and just the weight matrix $\boldsymbol{\Theta}$ is determined during the learning process.

Instantiating the single-layer neural network with the proposed ExpGC filter, we obtain as final model

$$\mathbf{Y} = \mathrm{softmax}\left(\sum_{i=0}^{k} \frac{\beta^i}{i!}\mathbf{L}^i\mathbf{X}\boldsymbol{\Theta}\right). \tag{13}$$

B. Linear Graph Convolution

By considering the ExpGC formulation in (13), we observe that the single parameter β determines the weights assigned to all the components in the summation (via the exponential series expansion). In this way, possible graph convolutions in the network are limited to those described by exponential filters. In particular, these filters act as low-pass filters that emphasize the contributions of the low-order monomials in the representation of $\mathbf{H}$ more than the high-order ones. This might be a restriction for some applications.

To allow larger families of polynomial filters in a network layer and to increase the expressive power of the convolution operator, we therefore study in this section also LGCs. The main idea is to replace the terms $(\beta^i/i!)$ with learnable parameters, one for each i, obtaining the following general formulation for a single layer:

$$\mathbf{H} = \sum_{i=0}^{k} \alpha_i \mathbf{L}^i\mathbf{X}\boldsymbol{\Theta}. \tag{14}$$

Similarly as before, we can define a single-layer LGC neural network as

$$\mathbf{Y} = \mathrm{softmax}\left(\sum_{i=0}^{k} \alpha_i\mathbf{L}^i\mathbf{X}\boldsymbol{\Theta}\right). \tag{15}$$

Note that in this work we derived the ExpGC and LGC formulation in terms of spectral filters based on the graph Laplacian $\mathbf{L}$, while the SGC operator can be written as a monomial of the perturbed operator $\mathbf{S}$. In fact, using $\mathbf{S}$ instead of $\mathbf{L}$ and fixing the coefficients $\alpha_0 = \cdots = \alpha_{k-1} = 0$, and $\alpha_k = 1$, we see that the SGC operator fits as well in this more general LGC framework, if we ignore the slightly different normalization for $\mathbf{S}$. Compared with the fixed SGC scheme, and the ExpGC scheme with one additional parameter β, the more flexible LGC formulation allows to learn $k+1$ coefficients for the convolution in the network layer.

This added expressiveness does not allow us to bound the approximation error introduced with respect to the version with $k = \infty$. In fact, having no constraints on the α_i parameters, each term of the summation can potentially significantly contribute to the final representation.

C. Rademacher Complexity for SGC, ExpGC, and LGC

The simple single-layered structure of the SGC, ExpGC, and LGC networks allows to obtain explicit estimates for their Rademacher complexity, a measure for the learnability of function classes in the respective networks. While bounds of the Rademacher complexity have been proven for deep GCNs with particular graph topologies [10], the simplicity of our networks allows for slightly tighter bounds that hold true for general graphs.

For a set $\mathcal{F}$ of real-valued signals $f : V \to \mathbb{R}$ on the graph and a sampling set $U = \{u_1, \ldots, u_L\} \subset V$ (usually i.i.d. random nodes distributed according to a given probability measure on the domain V), the empirical Rademacher complexity of $\mathcal{F}$ with respect to U is defined as follows:

$$\hat{\mathcal{R}}(\mathcal{F}) := \mathbb{E}_\epsilon\left[\sup_{f\in\mathcal{F}} \frac{1}{L}\sum_{\ell=1}^{L} \epsilon_\ell f(u_\ell)\right]$$

where $\mathbb{E}_\epsilon$ denotes the expectation with respect to a uniform distribution of $\epsilon \in \{-1,1\}^L$. Note that in the literature there exists a second variant of the Rademacher complexity, the so-called transductive Rademacher complexity [11]. Compared with the empirical setting used in this work, the transductive complexity also takes the nonsampled nodes into account.

Theorem 1: Let $a, b > 0$, $\sup_{j,j'} |\mathbf{X}_{j,j'}| \le M$, and consider the set

$$\mathcal{F}_{\mathrm{LGC}} = \left\{\mathbf{Y} = \sigma\left(\sum_{i=0}^{k} \alpha_i\mathbf{L}^i\mathbf{X}\boldsymbol{\Theta}\right) \mid \|\alpha\|_\infty \le a,\ \|\boldsymbol{\Theta}\|_1 \le b\right\}$$

where σ is any Lipschitz-continuous activation function with Lipschitz constants Λ and $\Theta \in \mathbb{R}^c$. Then the Rademacher complexity of $\mathcal{F}_{\mathrm{LGC}}$ with respect to any sampling set $U \subset V$ of size L is bounded by

$$\hat{\mathcal{R}}(\mathcal{F}_{\mathrm{LGC}}) \le \frac{abM\Lambda}{\sqrt{L}}\left(\frac{\|\mathbf{L}\|_1^{k+1} - 1}{\|\mathbf{L}\|_1 - 1}\right). \tag{16}$$

Proof: Without loss of generality, we can assume that $u_1 = v_1, \ldots, u_L = v_L$. Since the activation function σ is Lipschitz with constant $\Lambda > 0$, we can use the contraction property of the Rademacher complexity and obtain

$$\begin{aligned}\hat{\mathcal{R}}(\mathcal{F}_{\mathrm{LGC}}) &\le \frac{\Lambda}{L}\mathbb{E}_\epsilon\left[\sup_{\|\alpha\|_\infty\le a}\ \sup_{\|\boldsymbol{\Theta}\|_1\le b}\sum_{\ell=1}^{L}\epsilon_\ell\left(\sum_{i=0}^{k}\alpha_i\mathbf{L}^i\mathbf{X}\boldsymbol{\Theta}\right)_\ell\right]\\ &\le \sup_{\|\alpha\|_\infty\le a}\ \sup_{\|\boldsymbol{\Theta}\|_1\le b}\frac{\Lambda}{L}\left\|\left(\sum_{i=0}^{k}\alpha_i\mathbf{L}^i\right)\mathbf{X}\boldsymbol{\Theta}\right\|_\infty \mathbb{E}_\epsilon\left|\sum_{\ell=1}^{L}\epsilon_\ell\right|\\ &\le \frac{bM\Lambda}{L}\left(\sum_{i=0}^{k} a\|\mathbf{L}\|_1^i\right)\sqrt{L}\\ &= \frac{abM\Lambda}{\sqrt{L}}\left(\frac{\|\mathbf{L}\|_1^{k+1}-1}{\|\mathbf{L}\|_1-1}\right).\end{aligned}$$

Here, in the last step we used Jensen's inequality. □

As expected, this theoretical estimate indicates that the bound on the Rademacher complexity of the LGC increases as soon as the degree k of the polynomial kernel or the length parameters a and b in $\mathcal{F}_{\text{LGC}}$ increase. Similarly as in Theorem 1, we get the following estimates for the Rademacher complexities of the SGC and ExpGC networks.

Theorem 2: Let $b, d > 0$, $\sup_{j,j'} |\mathbf{X}_{j,j'}| \le M$, and consider the sets

$$\mathcal{F}_{\text{ExpGC}} = \left\{ \mathbf{Y} = \sigma\left(\sum_{i=0}^{k} \frac{\beta^i}{i!} \mathbf{L}^i \mathbf{X}\mathbf{\Theta} \right) \mid |\beta| \le d,\ \|\mathbf{\Theta}\|_1 \le b \right\}$$
$$\mathcal{F}_{\text{SGC}} = \left\{ \mathbf{Y} = \sigma\left(\mathbf{S}^k \mathbf{X}\mathbf{\Theta}\right) \mid \|\mathbf{\Theta}\|_1 \le b \right\}$$

where σ is any Lipschitz-continuous activation function with Lipschitz constants Λ and $\Theta \in \mathbb{R}^c$. Then the respective Rademacher complexities with respect to any sampling set $U \subset V$ of size L are bounded by

$$\hat{\mathcal{R}}\left(\mathcal{F}_{\text{ExpGC}}\right) \le \frac{bM\Lambda}{\sqrt{L}} e^{d\|\mathbf{L}\|_1}, \quad \hat{\mathcal{R}}(\mathcal{F}_{\text{SGC}}) \le \frac{bM\Lambda}{\sqrt{L}} \|\mathbf{S}\|_1^k. \quad (17)$$

For proper choices of a and d and using the matrix $\mathbf{L}$ in SGC instead of $\mathbf{S}$, the sets $\mathcal{F}_{\text{ExpGC}}$ and $\mathcal{F}_{\text{SGC}}$ are subsets of $\mathcal{F}_{\text{LGC}}$. In this case, the Rademacher complexities of $\mathcal{F}_{\text{SGC}}$ and $\mathcal{F}_{\text{ExpGC}}$ are per definition smaller than the Rademacher complexity of the larger set $\mathcal{F}_{\text{LGC}}$. This is also visible in the respective Rademacher bounds in (16) and (17), where for properly chosen lengths a and d, the bound in (16) dominates the bounds in (17). While the bound for $\hat{\mathcal{R}}(\mathcal{F}_{\text{SGC}})$ depends on the polynomial degree k, it is interesting that the bound for $\hat{\mathcal{R}}(\mathcal{F}_{\text{ExpGC}})$ is independent of k and depends rather on the range of the exponential rate β. In particular, if the norm of $\mathbf{S}$ is precisely 1, the Rademacher bound for $\hat{\mathcal{R}}(\mathcal{F}_{\text{SGC}})$ remains constant for increasing k, while we can expect that the complexity of $\mathcal{F}_{\text{ExpGC}}$ gets larger for increasing d.

Due to the simple one-layer structure of our networks, we got explicit upper bounds for their Rademacher complexities. This is not necessarily the case for more complex architectures where two or more layers are stacked. For these type of networks, it gets more difficult to derive meaningful bounds since a bound on a single layer has to be combinatorially reused leading in general to pessimistic upper bounds. For GCNs with one hidden layer, such tight upper bounds have been found in [12]. Tight upper bounds (with an additional logarithmic factor in L) for deep GCNs have been derived in [10] for specific graph topologies.

D. Expressive Power of SGC, ExpGC, and LGC

The Rademacher complexity analysis given in Section III-C provides us with a powerful tool to study the expressive power of the proposed graph convolutions. However, the available upper bounds for those complexities do not always allow to properly characterize the relationships among the different classes of functions that the applied convolutions are able to represent. In this section, we compare the expressive power of the three convolutions SGC, ExpGC, and LGC, in the sense of the family of functions that the convolutions in the single networks can represent. Let us refer as $\mathcal{H}_{\text{SGC}}$, $\mathcal{H}_{\text{ExpGC}}$, and $\mathcal{H}_{\text{LGC}}$ the family of functions that can be represented by SGC, ExpGC, and LGC, respectively. In the following, we will always use the graph Laplacian $\mathbf{L}$ instead of $\mathbf{S}$ in the SGC network.

Theorem 3: LGC in (15) is more expressive than SGC defined in (8).

Proof: Obviously, $\mathcal{H}_{\text{SGC}} \subseteq \mathcal{H}_{\text{LGC}}$ since for each function in SGC we can construct a corresponding LGC function where the indices α_i for the convolution filter are 0 for $i \neq k$ and 1 for $i = k$. The $\mathbf{\Theta}$ parameters remain the same.

It is also easy to see that $\mathcal{H}_{SGC} \neq \mathcal{H}_{\text{LGC}}$ since any function in $\mathcal{H}_{\text{LGC}}$ that has more than one coefficient α_i different from 0 can in general not be represented by SGC. □

Theorem 4: LGC in (15) is more expressive than ExpGC defined in (13).

Proof: It is easy to see that $\mathcal{H}_{\text{ExpGC}} \subseteq \mathcal{H}_{\text{LGC}}$ since, fixed any value of the parameter β in ExpGC, we can construct an equivalent LGC model by setting $\alpha_i = (\beta^i / i!)$.

To see that $\mathcal{H}_{\text{LGC}}$ is strictly larger than $\mathcal{H}_{\text{ExpGC}}$, we provide an example of a family of functions in $\mathcal{H}_{\text{LGC}}$ which is not in $\mathcal{H}_{\text{ExpGC}}$. Such a family is, for instance, given by a function with constant coefficients, i.e., where $\alpha_i = \alpha_j$ for all i, $j \le k$. This function cannot be represented by ExpGC given the exponential nature of its coefficients. □

As for the relation between ExpGC and SGC, besides the trivial case where $k = 0$ in which the two convolutions are equivalent, for $k > 0$ the two sets of functions that can be represented are disjunct.

E. Hyper-ExpGC

In the ExpGC convolution, a single learned parameter β determines the weight of each $i \in \{0, \dots, k\}$ term of the summation. Since ExpGC is based on the coefficients of the exponential power series, the weighted sum tends to emphasize the contribution of a small subset of the considered terms (we analyze this phenomenon more in deep in Section V-F). Besides that, it is important to note that the expressiveness of the convolution is also limited due to the fact that the β parameter is shared between all the considered nodes. We can, however, start from this intuition and define a mechanism to overtake this limitation. To allow the model to adjust dynamically β based on the input node, we propose the hExpGC, which exploits a network $f([\mathbf{X}, \mathbf{L}^1\mathbf{X}, \dots, \mathbf{L}^k\mathbf{X}])$ generating a vector $\boldsymbol{\beta} = [\beta_v | v \in V]$ having a different parameter β_v for each node $v \in V$. This idea is inspired by the hypernetworks [13]. Hypernetworks were introduced in the context of an RNN and CNN that were used to generate the weights of a primary model solving the actual task. However, the idea of having one network to predict the weights of another was proposed earlier and has reemerged multiple times [14], [15], [16].

In this work, $f()$ is defined by a shallow neural network

$$\begin{aligned} f\left(\left[\mathbf{X}, \mathbf{L}^1\mathbf{X} \dots, \mathbf{L}^k\mathbf{X}\right]\right) &= \left[\mathbf{X}, \mathbf{L}^1\mathbf{X}, \dots, \mathbf{L}^k\mathbf{X}\right] \cdot \mathbf{W} + b \\ \mathbf{Y} = \text{softmax}&\left(\sum_{i=0}^{k} \frac{f\left(\left[\mathbf{X}, \mathbf{L}^1\mathbf{X}, \dots, \mathbf{L}^k\mathbf{X}\right]\right)^i}{i!} \mathbf{L}^i \mathbf{X}\mathbf{\Theta} \right) \quad (18) \end{aligned}$$

where $\mathbf{W} \in \mathbb{R}^{s \cdot k \times 1}$, and b is a scalar.

The hExpGC convolution presents another interesting feature indeed, i.e., given a node v, by analyzing the value of $(\beta_v^i / i!)$ for each $i \in \{0, \dots, k\}$, it is possible to identify which values of k turn out to be more relevant to solve the considered classification task. We empirically analyze this feature in Section V-H.

F. Hyper-LGC

LGC aimed at increasing the expressiveness of ExpGC exploiting multiple (i.e., $k+1$) weighting parameters compared with the single parameter β of ExpGC, c.f., (13) and (15). We can further increase the expressiveness of LGC by defining a convolution with one parameter for each node and each layer. This convolution can be defined as follows:

$$\mathbf{H} = \sum_{i=0}^{k} \boldsymbol{\alpha}_i \odot \mathbf{L}^i \mathbf{X} \boldsymbol{\Theta} \tag{19}$$

where $\boldsymbol{\alpha}_i \in \mathbb{R}^n$ is a vector of weights, one for each node and for each i. Such definition, while allowing each node to aggregate information coming from multiple terms of the sum in a different way, would probably result in overfitting due to the high number of parameters and the lack of any regularization mechanism such as weight sharing. Moreover, it would not be possible to evaluate out of sample nodes, due to the undefined value of $\boldsymbol{\alpha}_i$. Because of that, and similar to what we did for hExpGC, we propose to define the weight vectors $\boldsymbol{\alpha}_i$ as the output of a function implemented by a neural network. The number of parameters of such network will be lower with respect to the number of nodes, thus forcing the exploitation of locality and feature similarity in the graph domain. We propose to parameterize such function on the input features propagated via the diffusion operator at each i-hop step. Thus, the resulting function is a graph convolutional neural network itself. Moreover, we implement it using a gating mechanism that, for each node, modifies the *base* parameter α_i.

The hLGC is defined as follows:

$$\mathbf{Y} = \text{softmax}\left(\sum_{i=0}^{k} \left(\mathbf{L}^i \mathbf{X} \boldsymbol{\Theta} \odot \sigma\left(f_i\left(\mathbf{L}^i \mathbf{X}\right)\right) \alpha_i \right) \right) \tag{20}$$

where $f_i()$ is a deep neural network that exploits ReLU as activation hidden function and returns a multiplicative factor value for each node. In the following, we refer to $f_i()$ as the hyper model. Note that differently from the common hyper neural network previously proposed in literature, the hExpGC and hLGC use a simpler hyper model. Indeed, the adopted ones are simple networks, while in the hyper neural network framework, it is common to use as hypernetwork a model that has a similar structure than the primary model (e.g., the hyper LSTM and the hyper CNN proposed in [13]).

An interesting feature of the hypernetworks is that their particular structure allows to overtake the limitation imposed by the weight sharing mechanism. Our approach slightly differs from the typical hypernet mechanism. In fact, both hExpGC and hLGC use the hyper model $f()$ just to relax the weight sharing limitation. Indeed, they do not delegate the management of all the weights to the hyper network $f()$, while maintaining most of its weights shared among all the graph nodes. The models exploit the hyper models just to create a multiplicative factor that allows the rescaling of the components of the computed embedding.

G. Expressive Power of hLGC and hExpGC

Theorem 5: hLGC is more expressive than LGC

Proof: First, we have $\mathcal{H}_{\text{LGC}} \subseteq \mathcal{H}_{\text{hLGC}}$ since, fixed any value for the α_i parameters of LGC, we can define the corresponding vectors α_i of hLGC with all the entries at the same value.

To show that $\mathcal{H}_{\text{LGC}} \neq \mathcal{H}_{\text{hLGC}}$, we consider for simplicity only the case of a graph with a single node feature, i.e., where $\mathbf{x} \in \mathbb{R}^{n \times 1}$. In this case, $\theta \in \mathbb{R}$ is a single number for both LGC and hLGC. For LGC, we can define $\gamma_i = \alpha_i \theta$ and rewrite every f_{LGC} in LGC as follows:

$$f_{\text{LGC}} = \sum_{i=0}^{k} \gamma_i \mathbf{L}^i \mathbf{x}.$$

Thus, LGC can represent all the linear combinations of the vectors $\mathbf{L}^i \mathbf{x}$. If $k < n$, then the space spanned by these vectors forms a proper subspace of $\mathbb{R}^n$. Conversely, for f_{hLGC} in hLGC we obtain

$$f_{\text{hLGC}} = \sum_{i=0}^{k} \boldsymbol{\gamma}_i \odot \mathbf{L}^i \mathbf{x}$$

where each $\boldsymbol{\gamma}_i \in \mathbb{R}^n$, so hLGC is able to represent a much larger subset or even the entire space $\mathbb{R}^n$ (depending on the feature information x). □

Theorem 6: hExpGC is more expressive than ExpGC

Proof: The inclusion $\mathcal{H}_{\text{ExpGC}} \subseteq \mathcal{H}_{\text{hExpLGC}}$ follows directly as in Theorem 5. For simplicity, we consider again only single node features $\mathbf{x} \in \mathbb{R}^{n \times 1}$. Then, we can rewrite every function in ExpGC as

$$f_{\text{ExpGC}} = \sum_{i=0}^{k} \theta \frac{\beta^i}{i!} \mathbf{L}^i \mathbf{x}$$

where θ is a single multiplicative parameter. The contribution of the coefficient $(\beta^i / i!)$ to the vector $\mathbf{L}^i \mathbf{x}$ is the same for each node. On the other hand, hExpGC allows to have a different coefficient $(\beta^i / i!)$ for each node. Thus, it can represent a richer set of functions of the form

$$f_{\text{hExpGC}} = \sum_{i=0}^{k} \theta \frac{\boldsymbol{\beta}^i}{i!} \odot \mathbf{L}^i \mathbf{x}.$$

In particular, each node can have associated a different (exponential) combination of the basis vectors $\mathbf{L}^i \mathbf{x}$. □

Theorem 7: hLGC is more expressive than hExpGC

Proof: This proof is very similar to the one provided in Theorem 4. □

Both hExpGC and hLGC can represent a wide range of functions. Of course, the definition of the hypernetwork providing in output the coefficients may (and should) limit such expressiveness and trade-off bias for variance.

H. Computational Complexity

SGC in (8) is very efficient compared with other convolutions based on message-passing, e.g., GCN in (5), because it is possible to precompute the terms $\mathbf{S}^k\mathbf{X}$. Considering our four proposed models in (13), (15), (18), and (20), we can note that for all of them the terms $\mathbf{L}^i\mathbf{X}$ can be precomputed as well. Thus, the computational requirements of our proposed convolutions are comparable to the ones of SGC. While the asymptotic complexity of SGC, of our proposed methods, and other convolutions based on message passing is the same, in practice SGC as well as our proposed convolutions can be significantly faster compared with, for instance, the very popular GCN (see Section V-D).

IV. Related Works

In the past few years, several models inspired by the graph convolution idea have been proposed. We already discussed some methods that are closer to our formulation in Section II. In this section, we detail other relevant methods in literature. The first definition of neural network for graphs (NN4Gs) dates back to the nineties [17]. More recently, NN4Gs [18] proposed the idea that has been rebranded later as *graph convolution*. Scarselli [2] defined a recurrent NN4Gs, which was later extended [19] removing the constraint for the recurrent system to be a contraction mapping. Duvenaud et al. [20] proposed a hierarchical approach similar to NN4G and inspired by circular fingerprints in chemical structures that, differently from NN4G, exploited backpropagation for training, later extended to consider edge labels by ECC [21]. Zhang et al. [22] proposed a propagation scheme for vertices' representations based on the random-walk graph Laplacian, similar to the one presented in (5), which has been extended to consider neighbors at multiple distances [23]. PATCHY-SAN [24] defines a different convolutions on graphs, which is conceptually closer to convolutions defined over images, exploiting a canonical ordering on graph vertices. Graph attention networks (GATs) [25] exploit a different convolution operator based on masked self-attention. The idea is to replace the adjacency matrix in the convolution with a matrix of attention weights. The authors propose to use multihead attention to stabilize the training. While it may be more complex to train, GAT allows to weight differently the neighbors of a node, and thus, it is a very expressive graph convolution. Fast GCN [26] uses node sampling to define a fast convolution operator, suited for the inductive setting. LNet and AdaLNet [27] exploit filters learned on an approximation of the Laplacian matrix. In particular, the model proposed in [27] exploits particular localized polynomial filters based on the Lanczos algorithm, which leverages multiscale information. Deep graph infomax (DGI) [28] trains a GCN in an unsupervised setting to obtain general node embeddings. GNN with auto-regressive moving average (ARMA) filters (ARMA) [29] defines an ARMA filter for graph convolution. In [30], an extension of GCN using random learning techniques and a least-squares regularization is investigated. This extension allows to speed up the learning phase and improves the performance of GCN in large-scale settings. In [31], a graph convolution based on the Haar transform has been proposed for GCNs to substitute the eigenbasis of the graph Laplacian in (1).

The convolutional operators in the models described above differ significantly from the ones considered in our analysis since they are developed with the aim to be part of a deeper model (excluding [30] where the model differs by the introduction of random weights and a regularization parameter). Indeed, they were developed to consider only the closest (1-hop far) neighboring nodes. Stacking several convolutional layers makes it possible to increase the considered receptive field. Moreover, to be effective the models that exploit this kind of operators tend to use nonlinearities. Our study is inspired by the SGC model that was developed pursuing a different direction: reduce the model complexity by removing nonlinearities and collapsing weight matrices between consecutive layers. In this work, we propose a convolutional operator (ExpGC) that is simpler to interpret than SGC and three others (LGC, hExpGC, and hyper-GC) that gradually increase the complexity of the model to improve the expressiveness of the operator. It is worth to note that all the GC operators considered in our analysis exploit a structure similar to the one exploited by the SGC and follow the same idea to remove complexities given in deep GNNs.

Recently, many works propose the idea of extending graph convolution layers to increase the receptive field size. In general, these models exploit the power series of the diffusion operator. Differently from the models proposed in this article (and the SGC), these models usually concatenate polynomial powers of the adjacency matrix. This kind of multiscale representation of the graph features is also known as graph-augmented multilayer perceptrons (GA-MLPs) [32]. Note that GA-MLPs exploit MLPs to aggregate the various elements of the power series of the adjacency matrix. That makes the GA-MLPs more complex (and more expressive) than the models considered in our analysis. As the focus of this article is on defining, analyzing, and comparing simple single-layer models, we consider GA-MLPs and the other models discussed in the following to go beyond the purpose of our analysis since they exploit more sophisticated, multilayer architectures compared with the ones we investigated.

One of the methods that rely on the multiscale representation idea is the diffusion CNN (DCNN) [33]. DCNN defines a different graph convolution (i.e., diffusion-convolution) that incorporates in the definition of graph convolution the diffusion operator, i.e., the multiplication of the input representation with a power series of the degree-normalized transition matrix. The model proposed in [34] instead of using neighbor aggregation function adopts graph augmented features that combine node degree features and multiscale graph propagated features. Moreover, the model aggregates the graph augmented features of each vertex and projects each of these subsets using an MLP. Luan et al. [35] propose to use the Krylov blocks to define two novel deep GCN architectures. The first one, named snowball, stacks several GC layers that concatenate multiscale features incrementally, resulting in a densely connected graph network. The second model, called truncated Krylov, concatenates multiscale features in each layer, in a way that the topological features from all the levels are mixed together. The MixHop graph convolution layer was proposed in [36], where a multilayer architecture based on such convolution is defined. Each layer of the model mixes a subset (managed as a hyperparameter) 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. Nt and Maehara [37] observed that multiplying graph signals with propagation matrices corresponds to low-pass filtering. Based on this observation, the authors proposed the graph filter neural network, which filters the input features by multiplication with graph filter matrices. An alternative multiresolution architecture, dubbed scalable inception graph neural networks (SIGNs), is proposed by Rossi et al. [38]. The most interesting feature of this architecture is its ability to scale GNN to a very large graph. The author proposes to exploit as building block for the model a set of exponentiations of linear diffusion operators. Then, in each building block a learnable matrix is used to linearly project every exponentiation of the diffusion operator. As reported by the authors, the SIGN layer is able to replicate some popular graph convolutional layers. To do this, SIGN exploits a learnable weight matrix, with dimensions depending on the number of considered degrees of the power series of the diffusion operator. That makes this model more complex to train than the ones here considered in our analysis. Moreover, SIGN layers exploit a nonlinear activation function. It is worth noting that a single layer of SIGN could be used to implement the SGC, ExpGC, and LGC operators, while SIGN cannot implement neither hExpGC nor hLGC. Yu et al. [39] recently proposed an efficient and scalable extension of the SIGN model developed to handle prediction problems on heterogeneous graphs.

Liu et al. [40] proposed a model to learn node representations by adaptively incorporating information from large receptive fields. The model computes the node feature transformation by exploiting an MLP network. The computed node features are then used to construct a multiscale representation exploiting the exponentiation of the adjacency matrix obtained from all the levels mixed together. More recently, a graph multiscale representation was exploited to define a linear graph operator, the polynomial GCN (PGCN) [41]. Even though in Chen et al. [32] show multiscale architectures not to be more expressive compared with GNNs, the authors proved PGCN to be more expressive than the most common convolution operators and their linear stacking. It is also worth noting that PGCN is proven to be a generalization of the most common spatially localized graph convolutions. Indeed, the authors show that common graph convolution operators (including SGC, ExpGC, and LGC) can be defined as specific instances of a single PGC. Similar to SIGN, however, it cannot implement the hyper-GCs considered in our analysis. In [42] and [43], the multiscale representation approach was used in conjunction with an attention/gating mechanisms resulting in scalable and efficient models, while in [44] it is used in conjunction with reservoir computing framework to obtain an extremely efficient GNN.

TABLE I
DATASETS STATISTICS. THE COLUMNS #TRAIN, #VAL, AND #TEST REPORT THE NUMBER OF NODES IN THE TRAINING, VALIDATION, AND TEST SETS, RESPECTIVELY

Dataset	#Classes	#Edges	#Train	#Val	#Test
Citeseer	6	9228	120	500	1000
Cora	7	10556	140	500	1000
Pubmed	3	88651	60	500	1000
WikiCS	10	216123	580	1769	5847
Ogbn-Arxiv	40	1166243	90941	29799	48603
Reddit	41	114848857	153431	23831	55703

V. RESULTS

In this section, we compare the proposed graph convolutional layers against several widely adopted models on six real-world node classification datasets.

A. Dataset

We empirically validated the proposed convolutions on six widely adopted datasets of node classification: Citeseer, Cora, PubMed, Reddit, WikiCS, and Ogbn-Arxiv. Each dataset is a graph, and in the first three of them, nodes represent documents and node features are sparse bag-of-words feature vectors. Specifically, in Citeseer, Cora, and PubMed the task requires to classify the research topics of papers. Each node represents a scientific publication described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from a dictionary. In the Reddit dataset, the task involves the classification of Reddit posts. Each node is a post, and the node label is the community, or "subreddit," that a post belongs to. The authors sampled 50 large communities and built a post-to-post graph, connecting posts if the same user comments on both. The WikiCS dataset [45] is a graph where the nodes represent Wikipedia Computer Science articles and the edges represent the hyperlinks among them. Also in this case the node features derive from the text, but differently from the other considered datasets, here they were calculated as the average of pretrained *GloVe* word embeddings instead of using binary bag-of-words vectors. In the Ogbn-Arxiv dataset [46], the nodes represent arXiv papers while a direct edge connects a paper cited by another one. Node labels consist of 128-D feature vectors obtained by averaging the embeddings of words in the title and abstract of the article. The task consists in classifying the subject areas of arXiv CS papers. Relevant statistics about the datasets are reported in Table I.

B. Experimental Setting and Implementation Details

We developed all the models involved in the comparison using deep graph library (DGL) [47]. As baseline models, we considered the SGC [see (8)], the GAT, and the GCN [see (5)] convolutions. For these models, we exploit the implementation provided by DGL. For all the datasets but Reddit, we solve the resulting optimization problem with the Adam algorithm (a variant of stochastic gradient descent with momentum and adaptive learning rate). For the Reddit dataset,

TABLE II
SETS OF HYPERPARAMETERS VALUES USED FOR MODEL SELECTION VIA GRID SEARCH

Dataset	learning rate	weight decay	drop out	k	#hidden
Citeseer	0.2, 0.02, 0.001	10^{-2}, $5\cdot10^{-3}$, $5\cdot10^{-4}$	0.0, 0.2, 0.5	2,5,10,20,40,50,60	4, 8, 16, 24
Cora	0.2, 0.05, 0.001	$5\cdot10^{-3}$,$5\cdot10^{-4}$,$5\cdot10^{-6}$,	0.0, 0.2, 0.5	2,5,10,20,40,60,80	4, 8, 16, 24
Pubmed	0.2, 0.05, 0.001	$5\cdot10^{-3}$,$5\cdot10^{-4}$,$5\cdot10^{-6}$,	0.0, 0.2, 0.5	2,5,10,20,40,60,80	4, 8, 16, 24
WikiCS	0.2, 0.05, 0.001	$5\cdot10^{-3}$,$5\cdot10^{-4}$,$5\cdot10^{-6}$	0.0, 0.2, 0.5, 0.7	2,5,10,20,50	4, 8, 16, 24, 48
Ogbn-Arxiv	0.2, 0.1, 0.05, 0.001	$5\cdot10^{-3}$,$5\cdot10^{-4}$,$5\cdot10^{-6}$, 0	0.0, 0.2, 0.5, 0.7	2,5,10,20,50	4, 8, 16, 24, 48
Reddit	1, 0.5, 0.05, 0.005	-	0.0,0.5	2,4,6	1, 2, 4, 8

TABLE III
ACCURACY COMPARISON ON NODE CLASSIFICATION TASK BETWEEN THE PROPOSED MODELS AND THREE BASELINES (SGC, GAT, AND GCN). THE MODEL SELECTION IS PREFORMED CONSIDERING THE RESULTS OBTAINED ON THE VALIDATION SET

Model \ Dataset	**Citeseer**	**Cora**	**Pubmed**	**WikiCS**	**Ogbn-Arxiv**	**Reddit**
SGC	69.2±0.0	80.1±0.04	79.8±0.0	79.5±0.31	64.6±0.36	94.7±0.05
GAT	70.7±0.81	80.5±1.02	78.3±0.95	79.1±0.27	**70.1±0.73**	OOM
GCN	71.3±0.48	81.0±0.67	79.2±0.63	79.4±0.52	65.9±3.26	90.1±0.12
ExpGC	71.3±0.15	80.3±0.46	79.4±0.22	80.1±0.63	64.6±0.62	94.5±0.01
LGC	72.2±0.33	82.0±0.52	80.6±0.40	80.0±0.45	65.9±0.92	95.2±0.06
hExpGC	71.5±0.80	82.3±1.05	80.0±0.11	78.4±0.35	51.6±0.66	94.8±0.08
hLGC	**72.3±0.10**	**82.4±0.88**	**80.8±0.14**	**80.1±0.47**	67.0±0.83	**95.6±0.07**

we use the limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm (L-BFGS) algorithm [48]. We used early stopping (with the patience set to 100) and model checkpoint, monitoring the accuracy on the validation set. We set the maximum number of epochs to 500. All the experiments involved a shallow model composed of a single layer followed by a softmax activation function. The results were obtained by performing five runs for each model. For our experiments, we adopted a machine equipped with: 2x Intel(R) Xeon(R) CPU E5-2630L v3, 192GB of RAM, and an Nvidia Tesla V100. For more details, check the publicly available code.[1]

1) Model Selection: Before discussing the results of the proposed graph convolutions in the perspective of results of the state-of-the-art methods, we would like to point out that for different reasons, the results reported in literature are not always comparable to the ones we report here. For instance, there may be different versions of the same dataset (using the same name), or different train/validation/test splits on the same dataset that may significantly impact the reported results. Another aspect to consider is the procedure adopted to select the hyperparameters (such as learning rate, regularization, and network architecture). Many papers report, for each dataset, the best performance on the test set obtained after testing many hyperparameter configurations. This procedure favors complex methods that depend on many hyperparameters, since they have a larger set of trials to select from compared with simpler methods. However, the predictive performances computed in this way are not unbiased estimations of the true error, and thus, these results are not comparable to other model selection methods [49]. For these reasons, we consider in this article two experimental settings. In the first experimental setting, reported in the following, we select all the hyperparameters of each method on the validation set. We then evaluate the test set with a single model. In the second one, that we report in Appendix A, we used an approach followed by many works in literature, i.e., we report the performance of the *best* hyperparameter configuration for each dataset. As mentioned before, *these results shall be considered as an upper bound* on the predictive performance of the method.

To ensure a fair comparison using a correct model selection procedure, we ran new experiments for all the models considered in the empirical comparison. For this reason, some of the reported results do not coincide with the results reported in literature. The hyperparameters of the model (number of hidden units, learning rate, weight decay, k) were selected using a limited grid search, where the explored sets of values do change based on the considered dataset. We performed some preliminary tests to select the set of values taken into account for each hyperparameter. In Table II, we report the sets of hyperparameter values used for the grid search. To perform a fair comparison among the proposed models and the baselines, we used the same hyperparameters grid for all the models. As evaluation measure, we used the average accuracy computed on the validation set.

C. Experimental Results

Table III reports the results obtained validating all the hyperparameters on the validation set. For each method and dataset, we report the average accuracy and the standard deviation over five runs. For sake of completeness, we have also reported in Appendix A the results obtained selecting the best hyperparameter values on the test set. We recall that this hyperparameter selection procedure is biased, as discussed in the previous section.

Let us start considering the first proposed graph convolution: ExpGC. It shows competitive predictive performance, performing more than 2% better than SGC on Citeseer and

[1]https://github.com/lpasa/SimpleGraphConvolutionalNetworks

being comparable to GAT and GCN. On Cora, ExpGC performs comparably to SGC and GAT, and slightly worse than GCN. On the PubMed dataset, ExpGC performs slightly worse than SGC, but comparably to GCN and slightly better than GAT. On WikiCS, it achieves the best accuracy obtaining the same performance as hLGC (with little more standard deviation). On Ogbn-Arxiv, the ExpGC is comparable to the GCN and SGC, while the accuracy is significantly lower than the one achieved by GAT. On Reddit, ExpGC is comparable to SGC, which in turn performs better than GCN. We could not compute GAT on Reddit since even using the simplest possible model (with a single attention head) the memory requirements are higher than the 16 GB that are available on our GPU.

Considering LGC, it performs better than ExpGC in all the considered datasets but WikiCS, while LGC performs better than hExpGC on Citeseer and PubMed and WikiCS. Moreover, it outperforms the competing methods in literature in all the considered datasets except Ogbn-Arxiv, which is the only dataset where a more complex nonlinear model like GAT shows better performance. Note that both GAT and GCN baselines exploit a layer with a different number of hidden units compared with SGC and the proposed ExpGC, hExpGC, and LGC models, which directly compute the representation in the output space. Finally, the hLGC model achieves the best predictive performance in all the considered datasets except Ogbn-Arxiv, where it obtains the second higher result.

D. Computational Requirements

In Table IV, we report the average computational time required to perform a single epoch for the three considered convolutions from the literature and our proposals. We report, for each convolution and dataset, the computational times corresponding to the hyperparameters that provide the best predictive results. We report the average duration (and standard deviation) of all the training epochs, in milliseconds. We can note that GAT and GCN are, in all the datasets, significantly slower compared with ExpGC, hExpGC, and LGC that allow to precompute the exponentiations of the adjacency matrix. This is not the case for hLGC due to the overhead introduced by the hypernetworks used for each i. It is worth notng that the hLGC scales very well on huge dataset like Ogb-Arxiv. Indeed, comparing the average computational time required to perform a single epoch with the only model that achieves a better accuracy (GAT) shows significantly lower time demands. Note that the reported times are an average over all the epochs, and thus, the preprocessing time for SGC, ExpGC, LGC, hExpGC, and hLGC is included in the reported time. In Table IV, we also report the number of learnable parameters of the models selected via grid search. The table shows that ExpGC, SGC, and LGC exploit a low number of parameters. In comparison to the GCN and GAT, the difference is in many cases of one order of magnitude. Moreover, it is worth noting that the hExpGC model uses a number of parameters that are similar to LGC, while the difference is considerably bigger if we consider hLGC. Even if both these models use hyper neural networks to generate the parameters of the convolution, the particular structure of hExpGC leads to a limit on the number of parameters that have to be tuned. It is also interesting to note that the number of parameters of LGC and ExpGC is very similar, and both close to the minimum number of learnable parameters that a model using the original input features has to manage [4].

TABLE IV

TIME COMPARISON AMONG THE MODELS PROPOSED IN THIS ARTICLE AND THREE BASELINES (SGC, GAT, AND GCN). FOR EACH CONVOLUTION AND DATASET, WE REPORT THE AVERAGE DURATION (AND STANDARD DEVIATION) OF ALL THE TRAINING EPOCHS OF THE BEST PERFORMING MODEL. THE TIME MEASUREMENTS ARE REPORTED IN MILLISECONDS. MOREOVER, WE ALSO REPORT THE NUMBER OF TRAINABLE PARAMETERS OF THE MODELS

	Citeseer	**Cora**	**Pubmed**	**WikiCS**	**Ogbn-Arxiv**
SGC	12±0.9	3.6±0.1	3.3±0.1	2.9±0.1	8.1±0.8
(k)	(20)	(5)	(10)	(2)	(2)
(#par)	22, 218	10, 031	1, 500	3, 000	5, 120
GAT	20.0±.0.4	20.3±0.2	19.3±0.2	67.7±5.2	408.5±8.2
(#hidden)	(8)	(4)	(16)	(8)	(24)
(#par)	241, 874	96, 661	33, 800	1, 262	33, 336
GCN	20.4±0.4	20.2±0.2	19.4±0.4	26.9±1.1	156.0±1.31
(#hidden)	(24)	(16)	(16)	(24)	(48)
(#par)	59, 366	34, 591	12, 099	7, 522	21, 928
ExpGC	4.9±0.2	6.8±0.4	4.5±0.3	5.0±0.1	17.2±0.9
(k)	(10)	(20)	(20)	(50)	(50)
(#par)	22, 219	10, 032	1, 501	3, 001	5, 121
LGC	6.8±0.2	10.1±0.4	5.2±0.3	5.2±0.2	31.7±0.4
(k)	(10)	(80)	(20)	(20)	(2)
(#par)	22, 229	10, 052	1, 521	3, 021	5, 123
hExpGC	3.2±0.4	6.24±0.1	7.5±0.6	9.5±0.9	38.1±0.4
(k)	(5)	(10)	(10)	(10)	(50)
(#par)	44, 438	25, 796	7, 002	6, 302	11, 650
hLGC	52.4±4.3	50.8±0.3	35.1±0.8	41.0±0.2	75.1±1.8
(k)	(10)	(80)	(40)	(10)	(2)
(#par)	13, 760, 349	2, 074, 985	253, 501	501, 322	30, 086

E. Discussion

While the improvement of hLGC compared with the second best performing method (LGC) on each dataset seems marginal, it is worth to note that hLGC consistently performs better than other methods. While all four proposed methods perform consistently better than SGC, hLGC is the method showing the best predictive performance, while LGC exhibits the best trade-off among predictive performance and required computational time.

F. Comparison Among Simple Convolutions

In Fig. 2, we report the loss curves during training using SGC and the four convolutions proposed in this work (ExpGC, LGC, hExpGC, and hLGC) on the Cora dataset. On the other datasets, similar considerations can be drawn. The plots report the loss computed on the training, validation, and test sets. The curves refer to the hyperparameters that yield the best results for each method (whose results are reported in Table V). It is interesting to note how the number of epochs that the model requires to converge is related to the expressiveness of the considered convolution. In fact, for SGC, ExpGC, LGC,

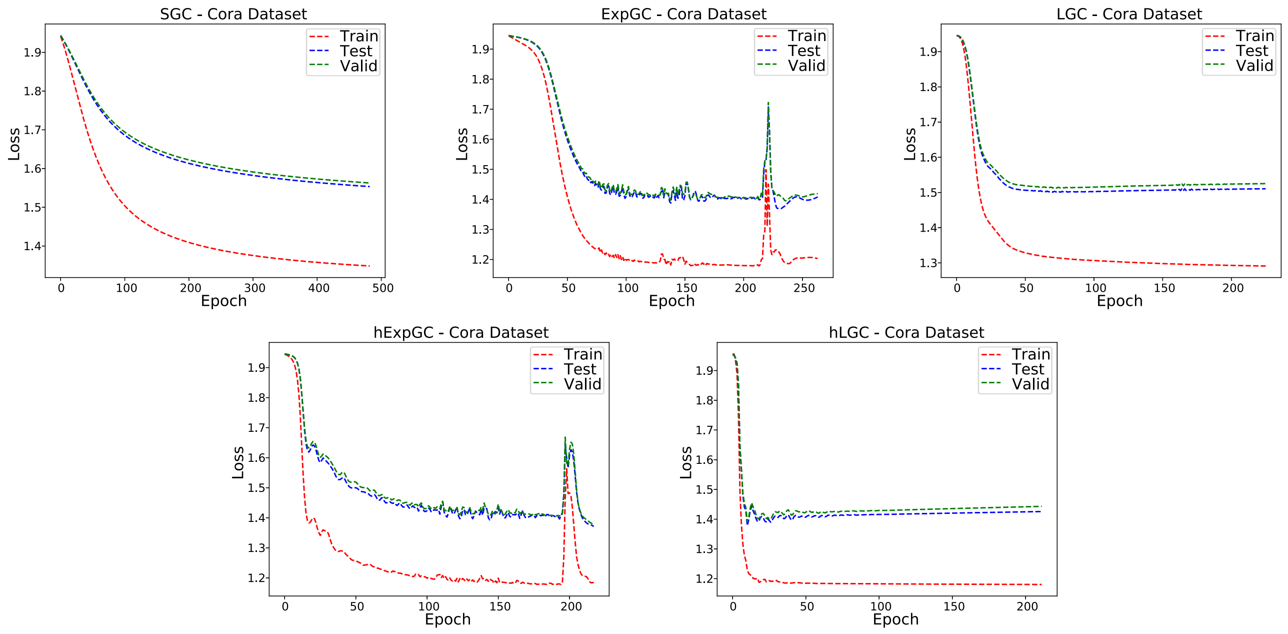


Fig. 2. Loss curves computed with the progress of training epochs on Cora (node classification) training, test, and validation sets for SGC, ExpGC, LGC, hExpGC, and hLGC.

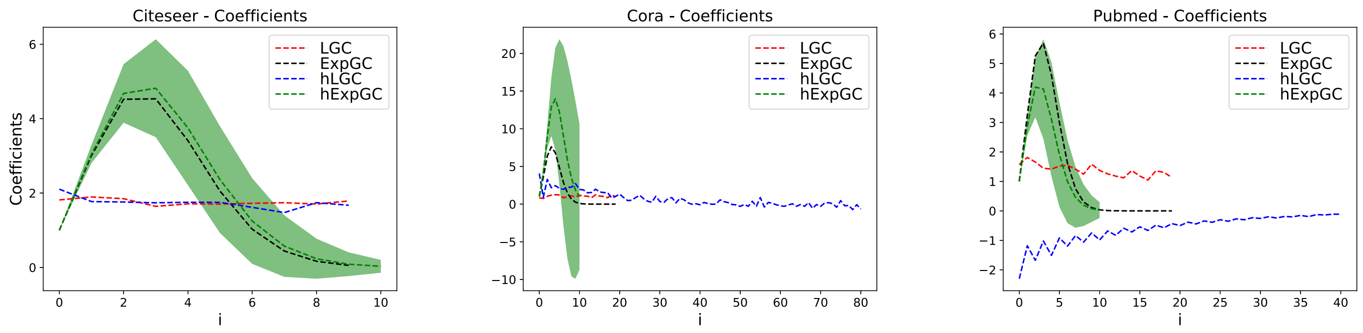


Fig. 3. Values of the α_i parameters of the LGC, values of $(\beta^i/i!)$ on ExpGC, (averaged) values of $(f([\mathbf{X}, \mathbf{L}^1\mathbf{X}, \ldots, \mathbf{L}^k\mathbf{X}])^i/i!)$ on hExpGC, and (averaged) values of $f_i(\mathbf{L}^i\mathbf{X})$ on hLGC (variance is reported as well, but in some cases it is too small to be visualized), for values of $i \in \{0, \ldots, k\}$. The considered models are trained on the Citeseer, Cora, and PubMed datasets and their hyperparameters are selected on the respective validation sets.

hExpGC, and hLGC (that are increasingly expressive), we can see that the slope of the curves becomes steeper as a more expressive model is used.

An important role in the three proposed convolutions is played by the multiplicative coefficients applied to each term of the summation, which significantly influence the optimization phase and the final results. For this reason, we decided to study the values of these multiplicative elements for all new models on the Citeseer, Cora, and PubMed datasets. In Fig. 3, we have reported the coefficient values for the three models selected in validation. For what concerns ExpGC, a single learned parameter β determines the weight of each $i \in \{0, \ldots, k\}$ term of the summation [see (13)], computed as $(\beta^i/i!)$. These values are represented by the black line in Fig. 3. For hExpGC, we report values of $(f([\mathbf{X}, \mathbf{L}^1\mathbf{X}, \ldots, \mathbf{L}^k\mathbf{X}])^i/i!)$, where $f([\mathbf{X}, \mathbf{L}^1\mathbf{X}, \ldots, \mathbf{L}^k\mathbf{X}])$ outputs a different value for each nodes. The LGC convolution defines, instead, a different multiplicative coefficient α_i for each $i \in \{0, \ldots, k\}$ (red line in Fig. 3). All α_i (similar to β) are adjusted during optimization. Finally, using the blue line, we report the average of the output of the hypernetworks $f_i(\mathbf{L}^i\mathbf{X})$ for each value of $i \in \{0, \ldots, k\}$ for the hLGC model. Variance is also reported; however, it is so small that it is not possible to discriminate it in the plot. We can note that the coefficients learned by LGC tend to be closer to each other compared with ExpGC, while the (average) coefficients generated in the hLGC show a much larger range of variation and diversification with respect to the other two models. The hExpGC learns coefficients that are similar to the ones learned by ExpGC, but with a considerable variance over nodes, thanks to the use of $f()$ that allows to have a different coefficient based on the features of the considered node in input. From these plots, it is evident that ExpGC is much more constrained with respect to LGC, being forced to concentrate significant values on few nearby terms. Moreover, the hLGC model selected in validation exploits a much larger value of k, thus showing a better ability to extract significant information from large receptive fields on the graph. Finally, the very small

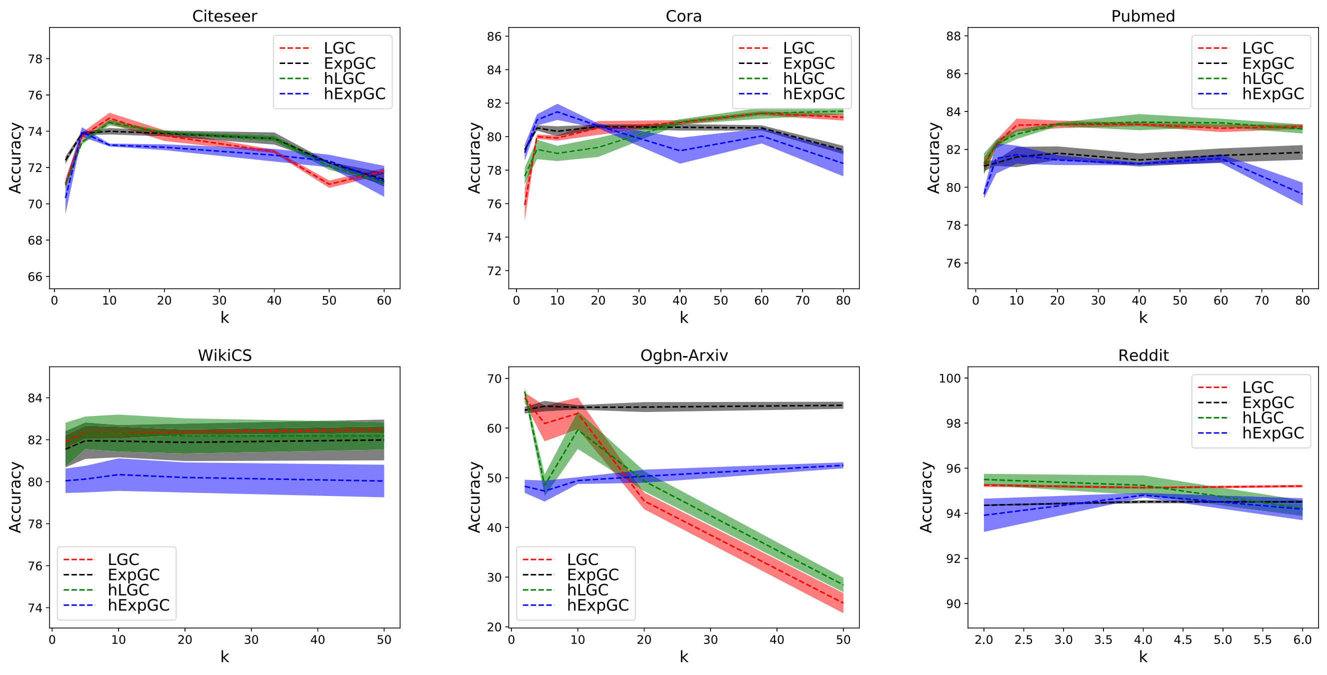


Fig. 4. Accuracy comparison of ExpGC, LGC, hExpGC, and hLGC using different values of k. The results are computed on the validation set of the datasets.

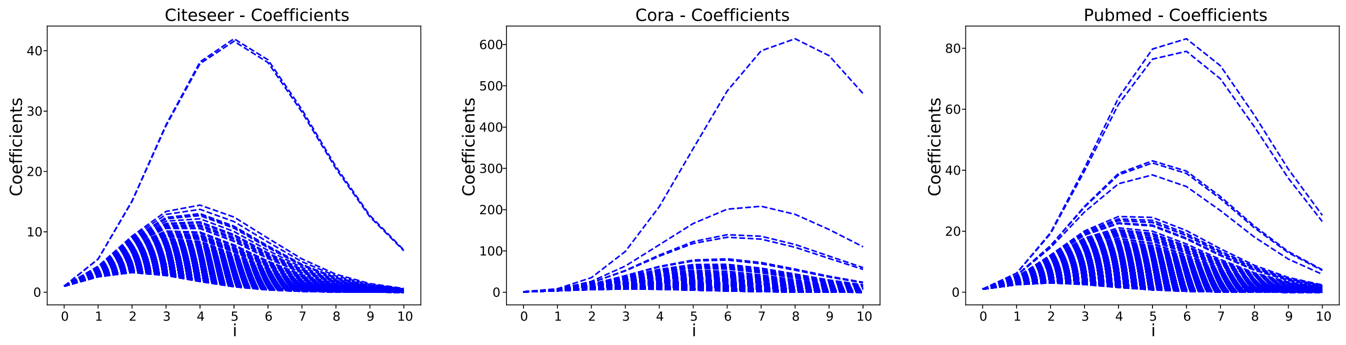


Fig. 5. Values of coefficients $(f([\mathbf{X}, \mathbf{L}^1\mathbf{X}, \ldots, \mathbf{L}^k\mathbf{X}])^i/i!)$ on hExpGC for each node in the Cora, Citeseer and PubMed datasets for values of $i \in \{0, \ldots, k\}$. The considered models are trained on the Citeseer, Cora, and PubMed datasets, and their hyperparameters are selected on the respective validation sets.

variance observed for the output of the hypernetworks seems to be an indication that the selected model does not overfit the training data.

G. Impact of Hyperparameter k

Our experimental results show that the selection of the best k value highly depends on the considered dataset. In Fig. 4, we analyze the accuracy achieved by the models proposed in this article, varying the hyperparameter k considering the values used during the validation phase (reported in Table II). The reported results are obtained using the validation set. In Citeseer, selecting smaller k values allows to obtain higher accuracy. Note that LGC and hLGC are more sensitive to this parameter. On Cora, the ExpCG and LGC-based models show different behaviors. Indeed, lower values of k are more suitable for ExpGC and hExpGC, while higher values are preferable for LGC and hLGC. On PubMed, all the models show stable performance using a value of $k \geq 5$. Similar stable trends can be seen on WikiCS and Reddit. It is worth noting that on PubMed, the hExpGC accuracy drops significantly using a too large a value for k. On Ogbn-Arxiv, k impacts significantly the performance of the LGC and hLGC. Indeed, the accuracy achieved by these models drops significantly for $k > 10$.

H. Interpretability of hExpGC

In this section, we study how the particular structure of hExpGC allows us to evaluate which are the most suitable i values that have to be considered to perform the node classification task. Indeed, the hyper model $f()$ (18) exploited by the hExpGC computes an ad hoc parameter β_v for each considered node v. The intuition is that the greater the value of $(\beta_v^i/i!)$, the higher the contribution of the embedding computed considering a receptive field of $i-hops$. In Fig. 5, each line represents the value of $(\beta_v^i/i!)$ for each $i \in \{0, \ldots, k\}$ of a single node contained in the dataset. It is interesting to note how each different node drives the hExpGC to focus on a different subset of the considered terms. The output of the hyper model $f()$also influences the size of the subsets of the

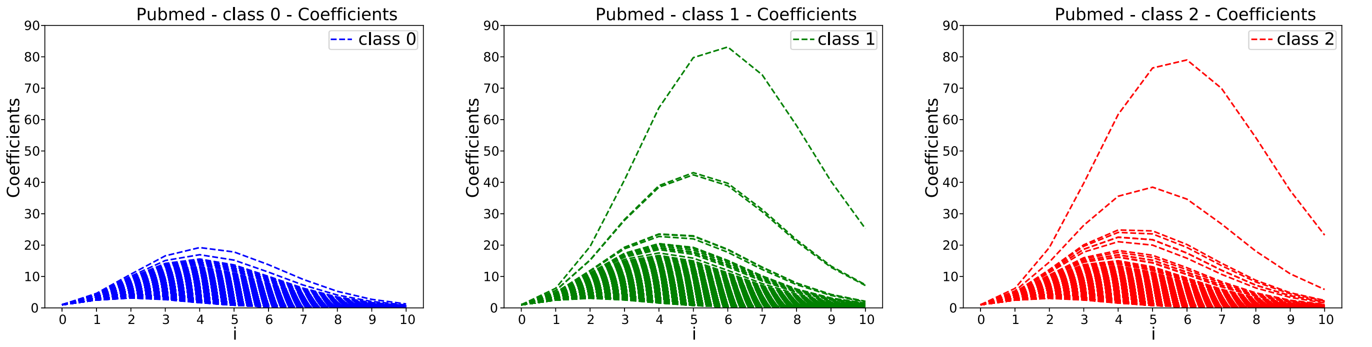


Fig. 6. Values of coefficients $(f([\mathbf{X}, \mathbf{L}^1\mathbf{X}, \ldots, \mathbf{L}^k\mathbf{X}])^i/i!)$ on hExpGC for each node of the PubMed dataset according to their target class for values of $i \in \{0, \ldots, k\}$. The considered model is obtained on the training set using hyperparameters values selected on the validation set.

TABLE V

ACCURACY COMPARISON BETWEEN THE PROPOSED MODELS AND THREE BASELINES (SGC, GAT, AND GCN). THE MODEL SELECTION IS PREFORMED CONSIDERING THE RESULTS OBTAINED ON THE TEST SET

Model\Dataset	**Citeseer**	**Cora**	**Pubmed**	**WikiCS**	**Ogbn-Arxiv**	**Reddit**
SGC	70.70±0.0	81.1±0.0	79.8±0.0	79.8±0.38	64.7±0.34	94.7±0.05
GAT	71.1±0.37	82.2±0.81	78.6±0.24	79.5±0.42	**70.3±0.10**	OOM
GCN	71.6±0.41	81.3±0.43	79.3±0.62	79.5±0.61	70.1±0.19	90.1±0.29
ExpGC	71.8±0.10	81.2±1.0	79.6±0.13	**80.5±0.47**	64.9±0.20	94.5±0.01
LGC	72.2±0.16	82.6±0.07	81.0±0.10	80.3±0.50	66.2±0.93	95.3±0.03
hExpGC	72.2±0.50	**83.5±0.44**	80.3±0.31	78.9±0.32	52.6±0.56	94.8±0.09
hLGC	**73.2±0.20**	83.0±0.01	**81.2±0.30**	**80.5±0.52**	67.3±0.73	**95.6±0.03**

considered terms. Indeed, for some nodes, many terms of the summation are multiplied by a factor close to zero. Studying how the values of the coefficients are distributed considering the various classes (targets) of the nodes, we note that there are no significant differences in the coefficient values' distribution among the nodes that belong to different classes. This suggests that coefficients obtained for each node are more influenced by node's embeddings than by the class it belongs to. An example of this behavior can be observed in Fig. 6, which reports the coefficient values distribution of the 3 classes of the PubMed dataset.

VI. CONCLUSION AND FUTURE DIRECTIONS

In this article, we followed the opposite direction compared with many works in literature on the definition of graph convolution operators. Instead of increasing the complexity of existing options, we started from the graph spectral filtering theory and defined four increasingly expressive graph convolutions. For two of these models, i.e., ExpGC and LGC, we also provided the Rademacher generalization bounds that, due to the simplicity of the proposed models, can be directly applied. We showed that our proposals achieve very good predictive performance while being more efficient (ExpGC, LGC, and hExpGC) to compute than most alternatives in the literature.

In the future, we plan to expand the study of the Rademacher complexity bounds on other graph convolutions and analyze whether the bounds are tight enough to allow for a comparison of the expressiveness of different graph operators. Moreover, we would like to study the effects of stacking multiple graph convolution layers in an architecture that includes nonlinearities. Finally, we plan to test the proposed convolutions in the setting of graph classification, instead of node classification, considered in this article.

APPENDIX A
RESULTS—HYPERPARAMETER SELECTION ON TEST SET

The results reported in Table V were obtained selecting the best hyperparameter values on the test set. For each hyperparameter configuration, the model with highest validation accuracy was selected. We recall that this hyperparameter selection procedure is biased, as discussed in Section V-B1. We can note that the performance obtained in the validated setting in Table III is in general lower compared with the ones in Table V. Complex methods such as GAT tend to show a higher decrease in accuracy.

REFERENCES

[1] N. Navarin and A. Sperduti, "Approximated neighbours minhash graph node kernel," in *Proc. ESANN*, 2017, pp. 281–286.
[2] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 61–80, Jan. 2009.
[3] F. Wu, T. Zhang, A. H. de Souza, C. Fifty, T. Yu, and K. Q. Weinberger, "Simplifying graph convolutional networks," in *Proc. ICML*, Feb. 2019, pp. 6861–6871.
[4] L. Pasa, N. Navarin, and A. Sperduti, "Compact graph neural network models for node classification," in *Proc. 37th ACM/SIGAPP Symp. Appl. Comput.*, J. Hong, M. Bures, J. W. Park, and T. Cerny, Eds., Apr. 2022, pp. 592–599, doi: 10.1145/3477314.3507100.
[5] N. Navarin, W. Erb, L. Pasa, and A. Sperduti, "Linear graph convolutional networks," in *Proc. Eur. Symp. Artif. Neural Netw., Comput. Intell. Mach. Learn.*, 2020, pp. 151–156.

[6] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. Neural Inf. Process. Syst. (NIPS)*, Jun. 2016, pp. 1–9.

[7] D. K. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory," *Appl. Comput. Harmon. Anal.*, vol. 30, no. 2, pp. 129–150, Mar. 2011.

[8] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE Signal Process. Mag.*, vol. 30, no. 3, pp. 83–98, May 2013.

[9] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. ICLR*, 2017, pp. 1–14.

[10] V. K. Garg, S. Jegelka, and T. Jaakkola, "Generalization and representational limits of graph neural networks," in *Proc. 37th Int. Conf. Mach. Learn. (ICML)*, 2020, pp. 3419–3430.

[11] P. Esser, L. C. Vankadara, and D. Ghoshdastidar, "Learning theory can (sometimes) explain generalisation in graph neural networks," in *Advances in Neural Information Processing Systems*, vol. 34, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds. Red Hook, NY, USA: Curran Associates, 2021, pp. 27043–27056. [Online]. Available: https://proceedings.neurips.cc/paper/2021/file/e34376937c784505d9b4fcd980c2f1ce-Paper.pdf

[12] S. Lv, "Generalization bounds for graph convolutional neural networks via Rademacher complexity," 2021, *arXiv:2102.10234*.

[13] D. Ha, A. Dai, and Q. V. Le, "Hypernetworks," in *Proc. ICLR*, 2017, pp. 1–29.

[14] B. Klein, L. Wolf, and Y. Afek, "A dynamic convolutional layer for short range weather prediction," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 4840–4848.

[15] G. Riegler, S. Schulter, M. Ruther, and H. Bischof, "Conditioned regression models for non-blind single image super-resolution," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 522–530.

[16] X. Jia, B. De Brabandere, T. Tuytelaars, and L. Van Gool, "Dynamic filter networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 29, 2016, pp. 667–675.

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

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

[19] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," in *Proc. ICLR*, 2016, pp. 1–20.

[20] D. Duvenaud et al., "Convolutional networks on graphs for learning molecular fingerprints," in *Proc. NIPS*, Montreal, QC, Canada, 2015, pp. 2215–2223.

[21] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *Proc. CVPR*, Jul. 2017, pp. 3693–3702.

[22] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 1–8.

[23] D. V. Tran, N. Navarin, and A. Sperduti, "On filter size in graph convolutional networks," in *Proc. IEEE SSCI*, Bengaluru, India, Nov. 2018, pp. 1534–1541.

[24] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 2014–2023.

[25] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *Proc. ICLR*, 2018, pp. 1–12.

[26] J. Chen, T. Ma, and C. Xiao, "FastGCN: Fast learning with graph convolutional networks via importance sampling," in *Proc. ICLR*, 2018, pp. 1–15.

[27] R. Liao, Z. Zhao, R. Urtasun, and R. S. Zemel, "LanczosNet: Multi-scale deep graph convolutional networks," in *Proc. ICLR*, 2019, pp. 1–18.

[28] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," in *Proc. ICLR*, 2019, p. 4.

[29] F. M. Bianchi, D. Grattarola, L. Livi, and C. Alippi, "Graph neural networks with convolutional ARMA filters," 2019, *arXiv:1901.01343*.

[30] C. Huang, M. Li, F. Cao, H. Fujita, Z. Li, and X. Wu, "Are graph convolutional networks with random weights feasible?" *IEEE Trans. Pattern Anal. Mach. Intell.*, early access, Jun. 15, 2022, doi: 10.1109/TPAMI.2022.3183143.

[31] M. Li, Z. Ma, Y. G. Wang, and X. Zhuang, "Fast Haar transforms for graph neural networks," *Neural Netw.*, vol. 128, pp. 188–198, Aug. 2020.

[32] L. Chen, Z. Chen, and J. Bruna, "On graph neural networks versus graph-augmented MLPs," 2020, *arXiv:2010.15116*.

[33] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *Proc. NIPS*, 2016, pp. 1993–2001.

[34] T. Chen, S. Bian, and Y. Sun, "Are powerful graph neural nets necessary? A dissection on graph classification," 2019, *arXiv:1905.04579*.

[35] S. Luan, M. Zhao, X.-W. Chang, and D. Precup, "Break the ceiling: Stronger multi-scale deep graph convolutional networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 10945–10955.

[36] S. Abu-El-Haija et al., "MixHop: Higher-order graph convolutional architectures via sparsified neighborhood mixing," in *Proc. 36th Int. Conf. Mach. Learn. (ICML)*, vol. 97, K. Chaudhuri and R. Salakhutdinov, Eds., Long Beach, CA, USA, Jun. 2019, pp. 21–29. [Online]. Available: http://proceedings.mlr.press/v97/abu-el-haija19a.html

[37] N. T. Hoang and T. Maehara, "Revisiting graph neural networks: All we have is low-pass filters," 2019, *arXiv:1905.09550*.

[38] F. Frasca, E. Rossi, D. Eynard, B. Chamberlain, M. Bronstein, and F. Monti, "SIGN: Scalable inception graph neural networks," 2020, *arXiv:2004.11198*.

[39] L. Yu, J. Shen, J. Li, and A. Lerer, "Scalable graph neural networks for heterogeneous graphs," 2020, *arXiv:2011.09679*.

[40] M. Liu, H. Gao, and S. Ji, "Towards deeper graph neural networks," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2020, pp. 338–348.

[41] L. Pasa, N. Navarin, and A. Sperduti, "Polynomial-based graph convolutional neural networks for graph classification," *Mach. Learn.*, vol. 111, no. 4, pp. 1205–1237, Apr. 2022.

[42] W. Zhang et al., "Graph attention MLP with reliable label utilization," 2021, *arXiv:2108.10097*.

[43] L. Pasa, N. Navarin, and A. Sperduti, "Simple multi-resolution gated GNN," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Dec. 2021, pp. 1–7.

[44] L. Pasa, N. Navarin, and A. Sperduti, "Multiresolution reservoir graph neural network," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 6, pp. 2642–2653, Jun. 2022, doi: 10.1109/TNNLS.2021.3090503.

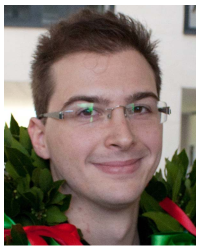
[45] P. Mernyei and C. Cangea, "Wiki-CS: A Wikipedia-based benchmark for graph neural networks," 2020, *arXiv:2007.02901*.

[46] W. Hu et al., "Open graph benchmark: Datasets for machine learning on graphs," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 22118–22133.

[47] M. Wang et al., "Deep graph library: A graph-centric, highly-performant package for graph neural networks," 2019, *arXiv:1909.01315*.

[48] R. H. Byrd, S. L. Hansen, J. Nocedal, and Y. Singer, "A stochastic quasi-Newton method for large-scale optimization," *SIAM J. Optim.*, vol. 26, no. 2, pp. 1008–1031, Apr. 2016.

[49] L. Oneto, *Model Selection and Error Estimation in a Nutshell*. Cham, Switzerland: Springer, 2020. [Online]. Available: https://link.springer.com/book/10.1007/978-3-030-24359-3



Luca Pasa received the B.Sc. and M.Sc. degrees in computer science from the University of Padua, Padua, Italy, in 2011 and 2013, respectively, and the Ph.D. degree in mathematical sciences (curriculum computer science) from the University of Padua, in 2017, under the supervision of Prof. Alessandro Sperduti.

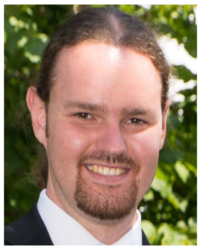
He was a Post-Doctoral Researcher at the Center for Translational Neurophysiology of Speech and Communication (CTNSC), the Italian Institute of Technology (IIT), and the Department of Mathematics, University of Padua. He is currently an Assistant Professor in computer science at the Department of Mathematics "Tullio Levi-Civita," University of Padua. He has coauthored many research papers published in international refereed journals and conference proceedings and he has been actively involved in the organization of several special sessions at international machine learning conferences. His research interests include automatic speech recognition, neural networks, and deep learning with a particular focus on sequential and structured domains.

Dr. Pasa is a member of IEEE Task Force on Deep learning and of the Italian Association for Artificial Intelligence.

Nicolò Navarin (Member, IEEE) received the Ph.D. degree in computer science from the University of Bologna, Bologna, Italy, in 2014.

He has been a Visiting Researcher at the University of Freiburg, Freiburg im Breisgau, Germany, and the Universitá della Svizzera Italiana, Lugano, Switzerland. He has been a Research Fellow at the University of Nottingham, Nottingham, U.K. and the University of Padua. He is a Tenuretrack Assistant Professor in computer science at the Department of Mathematics “Tullio Levi-Civita,” University of Padua, Padua, Italy. His research interests lie in the field of machine learning, including kernel methods and neural networks for structured data, online and continual learning, and trustworthy ML.

Dr. Navarin is a member of the IEEE Computational Intelligence Society, the IEEE Task Force on Deep Learning, and the IEEE Task Force on Learning from Structured Data. He has been serving as a PC member in major machine learning conferences, and he has been actively involved in the organization of several special sessions (ESANN, WCCI, and IJCNN) and conferences (INNS Big Data and Deep Learning 2019, International Conference on Process Mining 2020, IEEE Symposium Series in Computational Intelligence 2021, IEEE World Congress on Computational Intelligence 2022). He is an Associate Editor for the *Journals Evolving Systems* (Springer) and *Neurocomputing* (Elsevier), and an Editorial Board Member for Intelligenza Artificiale (AIxIA, IOS Press).



Wolfgang Erb received the Ph.D. degree in mathematics from the Technical University of Munich, Munich, Germany, in 2010.

He is an Assistant Professor in numerical analysis at the Department of Mathematics “Tullio Levi-Civita,” University of Padua, Padua, Italy. He has been a Research Fellow at the University of Lübeck, Lübeck, Germany, the University of Eichstätt-Ingolstadt, Eichstätt, Germany, and an Assistant Professor in mathematics at the University of Hawaii at Manoa, Honolulu, HI, USA. His research interests include multivariate approximation theory, kernel methods for signal processing and learning on graphs, fast and efficient reconstruction algorithms for inverse problems, and applications in biomedical imaging, in particular magnetic particle imaging.

Dr. Erb is a member of Italian Research Network on Approximation (RITA), the Italian Mathematical Society UMI (working group TAA of approximation theory), and the GNCS-INdAM.

Alessandro Sperduti (Senior Member, IEEE) received the Ph.D. degree from the University of Pisa, Pisa, Italy, in 1993.

He is a Full Professor at the Department of Mathematics of the University of Padua, Padua, Italy. Previously, he was an Assistant Professor (1995–1998) and an Associate Professor (1998–2002) at the Department of Computer Science, University of Pisa. He is the author of more than 220 publications on refereed journals, conferences, and chapters in books. His research interests are mainly in neural networks, kernel methods, and process mining.

Dr. Sperduti has been member of the European Neural Networks Society (ENNS) Executive Committee, the Chair of the DMTC of IEEE CIS for the years 2009 and 2010, the Chair of the NNTC for the years 2011 and 2012, the Chair of the IEEE CIS Student Games-Based Competition Committee for the years 2013 and 2014, and the Chair of the Continuous Education Committee of the IEEE Computational Intelligence Society for year 2015. He was a recipient of the 2000 AI*IA (Italian Association for Artificial Intelligence) “MARCO SOMALVICO” Young Researcher Award. He has been an invited plenary speaker in Neural Networks Conferences. He has served as AC in major AI conferences and currently is in the editorial board of the journals *Theoretical Computer Science (Section C)*, *Natural Computing*, and *Neural Networks*.