pathGCN: Learning General Graph Spatial Operators from Paths

Moshe Eliasof [1] **Eldad Haber** [2] **Eran Treister** [1]

Abstract

Graph Convolutional Networks (GCNs), similarly to Convolutional Neural Networks (CNNs), are typically based on two main operations - spatial and point-wise convolutions. In the context of GCNs, differently from CNNs, a pre-determined spatial operator based on the graph Laplacian is often chosen, allowing only the point-wise operations to be learnt. However, learning a meaningful spatial operator is critical for developing more expressive GCNs for improved performance. In this paper we propose pathGCN, a novel approach to learn the spatial operator from random paths on the graph. We analyze the convergence of our method and its difference from existing GCNs. Furthermore, we discuss several options of combining our learnt spatial operator with point-wise convolutions. Our extensive experiments on numerous datasets suggest that by properly learning both the spatial and point-wise convolutions, phenomena like over-smoothing can be inherently avoided, and new state-of-the-art performance is achieved.

1. Introduction

The study of Graph Convolutional Networks (GCNs) has gained large popularity in recent years (Bruna et al., 2013; Defferrard et al., 2016; Kipf & Welling, 2016; Bronstein et al., 2017; Monti et al., 2017) in a wide variety of fields and applications such as computer graphics and vision (Boscaini et al., 2016; Monti et al., 2017; Wang et al., 2018; Eliasof & Treister, 2020), Bioinformatics (Strokach et al., 2020; Jumper et al., 2021), node classification (Kipf & Welling, 2016; Chen et al., 2020; Chamberlain et al., 2021) and others. The common ingredient that most of the methods share

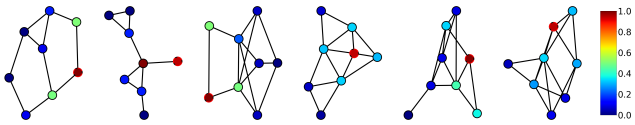

Figure 1. The spatial operator induced by a smoothing kernel on different graphs. The vertex with dashed outline is the path origin.

is the use of a pre-determined *spatial* operator, often times based on the graph Laplacian. While this choice is intuitive and effective, it induces limitations on the behaviour of GCNs. First, it is limited in the aspect of the expressiveness of the networks. Unlike CNNs (Krizhevsky et al., 2012; He et al., 2016; Howard et al., 2017), where both the spatial filters (e.g., 3×3 depth-wise convolutions) and point-wise (1×1) convolutions are learnt, here only the latter are left to be determined. Secondly, it is well known (Wu et al., 2019) that the Laplacian operator, when applied as in (Kipf & Welling, 2016) smooths the input features, and therefore a recurrent application of it may lead to over-smoothing, resulting in typically shallow networks. This phenomenon is well documented and studied in the field of GCNs (Wu et al., 2019; Zhao & Akoglu, 2020; Chen et al., 2020; Chamberlain et al., 2021; Eliasof et al., 2021). In this work we propose *pathGCN* – a novel approach that overcomes the limitations above, based on aggregation from random paths defined over the graph vertices. We show that using this approach it is possible to define spatial operators similarly to the ones used in 2D convolutions on images. Such operators have variable aperture and coefficients that may increase the expressiveness of GCNs. In addition, since the coefficients of the operator are learnt, its eigenvalues can be rather different than those of the graph Laplacian. This implies that the learnt kernels can take different roles, from smoothing to edge-detecting (or sharpening) operators. An example of the effective spatial operators that are induced by the smoothing spatial kernel $[0.8, 1.0, 0.6]$ is presented in Fig. 1, where we can see that the spatial operator is dependent both on the graph topology *and* the spatial kernel.

We provide an analysis of our method to motivate our approach in Sec. 3, and present actual learnt kernels by our network in Fig. 4, which suggests that greater spatial flexibility is required to obtain better performance and to avoid over-smoothing, as reflected in our experiments in Sec. 4.

[1]Department of Computer Science, Ben-Gurion University, Israel. [2]Department of Earth, Ocean and Atmospheric Sciences, University of British Columbia, Canada.. Correspondence to: Moshe Eliasof <eliasof@post.bgu.ac.il>, Eran Treister <eran@cs.bgu.ac.il>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

Our contributions are as follows:

- We introduce *pathGCN* – a novel approach for learning expressive spatial operators for GCNs from random paths. pathGCN supports several formulations, similarly to standard CNNs – ranging from a global to per layer and per channel learnt spatial operators.
- We provide an analysis of the behaviour of our pathGCN, and present a stochastic path training policy.
- Our experiments reveal the significance of the learnt spatial operator by obtaining and improving the state-of-the-art accuracy on various benchmarks, while also inherently preventing over-smoothing.

2. Related work

2.1. Graph convolutional networks

Notations. Assume we are given an undirected graph defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is a set of n vertices (nodes) and $\mathcal{E}$ is a set of m edges. Let us denote by $\mathbf{f}_i \in \mathbb{R}^c$ the feature vector that resides at the i-th node of $\mathcal{G}$ with c being the number of channels. Also, we denote the adjacency matrix $\mathbf{A}$, where $\mathbf{A}_{ij} = 1$ if there exists an edge $(i, j) \in \mathcal{E}$, and the diagonal degree matrix $\mathbf{D}$ where $\mathbf{D}_{ii}$ equals to the degree of the i-th node. The graph Laplacian is given by $\mathbf{L} = \mathbf{D} - \mathbf{A}$, and its symmetric normalized formulation reads $\mathbf{L}^{sym} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ where $\mathbf{I}$ is the identity operator. Let us also denote the adjacency and degree matrices after adding a self-loop to the nodes by $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{D}}$ respectively, and accordingly define the normalized Laplacian of $\mathcal{G}$ with added self-loops by $\tilde{\mathbf{L}}^{sym} = \mathbf{I} - \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$. It follows that the spatial operation from GCN (Kipf & Welling, 2016), induced by the graph Laplacian is given by $\tilde{\mathbf{P}} = \mathbf{I} - \tilde{\mathbf{L}}^{sym}$. We refer the readers to (Wu et al., 2019) for more information.

The spatial operator in GCNs. GCNs typically involve two main ingredients: spatial and point-wise (1×1) convolutions to mix the channels. The majority of GCNs employ a spatial operator based on the graph Laplacian, followed by a point-wise convolution. For example, GCN (Kipf & Welling, 2016) is given by:

$$\mathbf{f}^{(l+1)} = \sigma(\mathbf{S}^{(l)}\mathbf{f}^{(l)}\mathbf{W}^{(l)}) \tag{1}$$

where $\mathbf{S}^{(l)} = \tilde{\mathbf{P}}$, and $\mathbf{W}^{(l)}$ is a 1×1 convolution operator. The combination of the operator $\tilde{\mathbf{P}}$ and a learnable 1×1 convolution continued in a series of works (Wu et al., 2019; Chen et al., 2020; Zhou et al., 2021). For instance, the spatial operation in GCNII (Chen et al., 2020) can be obtained by replacing $\mathbf{S}^{(l)}$ with :

$$\mathbf{S}^{(l)}(\mathbf{f}^{(l)}, \mathbf{f}^{(0)}) = (1 - \alpha^{(l)})\tilde{\mathbf{P}}\mathbf{f}^{(l)} + \alpha^{(l)}\mathbf{f}^{(0)}, \tag{2}$$

with $\alpha^{(l)} \in [0, 1]$ being a hyper-parameter and $\mathbf{f}^{(0)}$ are the features of the first (embedding) layer, which is also similar to APPNP (Klicpera et al., 2019). Another recent example is EGNN (Zhou et al., 2021) which performs

$$\mathbf{S}^{(l)}(\mathbf{f}^{(l)}, \mathbf{f}^{(0)}) = (1 - c_{min})\tilde{\mathbf{P}}\mathbf{f}^{(l)} + \alpha\mathbf{f}^{(l)} + \beta\mathbf{f}^{(0)}, \tag{3}$$

where $c_{min} = \alpha + \beta$, and α , β are learnt scalars. While the methods above achieve impressive accuracy, they are still limited in their spatial expressiveness due to their dependence on $\tilde{\mathbf{P}}$, which may lead to sub-optimal performance.

On the other hand, there are methods that allow the freedom of learning a rich spatial operator. For example, ChebNet (Defferrard et al., 2016) learns dense convolutions through polynomial parameterization of the graph Laplacian. While from a conceptual perspective, ChebNet should be able to learn diverse operators and prevent over-smoothing, it still exhibits a degradation in performance when adding more layers, as discussed in (Levie et al., 2018) and according to our experiments as presented in Fig. 3. In addition, GDC (Gasteiger et al., 2019) propose to impose constraints on the filters of ChebNet to obtain diffusion kernels. Another network to consider is MoNet (Monti et al., 2017), which learns local patch operators by a mixture of Gaussian. While the Gaussian parameterization can yield more expressive kernels and favourable results compared to GCN (Kipf & Welling, 2016), it is more computationally demanding and challenging to train as more layers are stacked.

2.2. Random walk on graphs

The concept of random walk is useful in many applications and domains, from graph node classification (Perozzi et al., 2014; Nikolentzos & Vazirgiannis, 2020) to RNA disease association (Lei & Bian, 2020) and mesh denoising (Sun et al., 2008). In the context of graphs in machine and deep learning, multiple methods utilized random walks for different purposes. DeepWalk (Perozzi et al., 2014) is a two-step algorithm for node embedding learning, based on random walks and the SkipGram (Mikolov et al., 2013) method. RWGNN (Nikolentzos & Vazirgiannis, 2020) learns a set of hidden graphs which are then compared with an input graph using a differentiable mutual random walk counting procedure. Our approach is different as we utilize the random walk to effectively sample paths, and employ them to learn a spatial operator by the means of a convolution kernel. Another difference is that RGWNN considers the graph classification problem, as it employs the random walk kernel which is a scalar binary function of two graphs, whose output discards the notion of the graph itself. Therefore, it is not straight-forward to use it for node classification tasks. The recent PAN (Ma et al., 2020) parameterizes the convolution kernel by a polynomial of the adjacency matrix, where the coefficients are the learnt parameters, motivated by path integral theory. However, as its formulation consists of

learning non-negative weights, it is prone to over-smoothing and reduced expressiveness. Another related work that uses random walk strategy is GraphSAGE (Hamilton et al., 2017) where neighbourhood sampling of k hops is performed. This method is based on two steps in which an aggregation from two subsequent hops is performed, followed by a 1×1 convolution. This is different than ours, as we first extract the complete path, and learn a convolution kernel based on the original information along this path, while the former iteratively aggregates from subsequent hops.

3. Method

3.1. Learning spatial operators

In this section we motivate the need for learnt spatial operators. To do that, let us first consider a standard CNN where the data resides on a simple uniform mesh-grid. We note that CNNs and GCNs both represent data with geometrical features. However, while CNNs are networks that operate on a simple mesh-grid graph where pixels (nodes) are linked based on their location, and the local geometry of the graph is fixed, GCNs can be thought of as unstructured meshes where the local geometry varies.

Given a feature tensor $\mathbf{f} \in \mathbb{R}^{n \times c}$, the convolution in CNNs denoted by $\mathbf{Kf}$, is a linear operation where each input channel and each output channel has its own spatial operator. Thus, each linear operator $\mathbf{K}$ consists of $c \times c$ different spatial convolutions represented by the tensor $\mathbf{K}$. Furthermore, the convolutions can have a variable aperture (i.e., kernel size) obtaining a larger field of view, and are typically learnt per-layer, yielding a highly expressive set of operators.

In contrast, many popular and recent GCNs (Kipf & Welling, 2016; Wu et al., 2019; Chen et al., 2020; Chamberlain et al., 2021; Zhou et al., 2021) and others employ a pre-determined spatial operator, often times guided by the graph Laplacian which is determined solely by the topology of the graph $\mathcal{G}$, coupled with a 1×1 convolution to mix the channels. Therefore, by comparing GCNs to CNNs it is notable that the former have significantly fewer degrees of freedom with respect to their spatial operation. Indeed, 2D CNNs with a spatial kernel of size k and c channels optimize $c \times c \times k \times k$ parameters, while GCNs typically only optimize the 1×1 convolution, yielding $c \times c$ parameters. In addition to the possible expressiveness issue, the frequent of the graph Laplacian can lead to undesired phenomena such as over-smoothing (Wu et al., 2019; Zhao & Akoglu, 2020; Chen et al., 2020). Some attempts to overcome the expressiveness limitation consider using polynomials of the graph Laplacian, stabilizing them by constructing a Chebyshev basis (Defferrard et al., 2016). However, it imposes high computational cost due to the frequent computations of the Chebyshev polynomial and fully-connected convolution filters. Furthermore, it is difficult to train such a network due to its eigenvalues distribution, as demonstrated in (Levie et al., 2018). With respect to the over-smoothing phenomenon, various techniques were proposed (Zhao & Akoglu, 2020; Chen et al., 2020; Rong et al., 2020). While those methods indeed aid the over-smoothing issue, they do not inherently change the smoothing behaviour of GCNs, but rather ease the smoothing process.

In what follows, we present a methodology that allows the construction of a graph convolution that is similar to the standard convolution on a regular mesh grid which allows greater expressiveness and inherently does not over-smooth.

3.2. From fixed to variable spatial operator

The non-constant topology of the graph is the major obstacle in generating a meaningful spatial convolution in GCNs. We now show that this can be addressed by using random walks.

To this end, we consider a *path* on the graph, on which the weights that parameterize the spatial operator are learnt. Therefore, a transition function that dictates some traversal strategy on the graph is required in order to obtain a path like input to our network. Specifically, we adopt the graph random walk generator from node2vec (Grover & Leskovec, 2016), as implemented in PyTorch-Geometric (Fey & Lenssen, 2019), for its simplicity and efficient implementation. We note, however, that a different transition function could also be used or learnt.

Assume first that we have **a single channel** feature tensor $\mathbf{f} \in \mathbb{R}^{n \times 1}$, and a path of length k is given. Let us denote the learnt spatial parameters by $\mathbf{s} \in \mathbb{R}^k$, and let $y_j = (j_0, ..., j_{k-1})$ be a tuple of node indices of a single random path of nodes of length k, starting from node $j_0 = j$. The convolution over *a single path* for the j-th node is defined by the following linear operator:

$$\mathbf{K}_{y_j}(\mathbf{s})\mathbf{f} = \sum_{i=0}^{k-1} s_i f_{j_i}. \tag{4}$$

That is, the features of the nodes on the path y_j are weighted by the corresponding learnt parameter in $\mathbf{s}$, and summed to get the feature of the j-th node—the node where the path originates from.

More generally, instead of considering a single path, let us sample p different paths, and accordingly define the paths convolution as the average over all sampled paths

$$\mathbf{K}_{\mathcal{Y}_j}(\mathbf{s})\mathbf{f} = \frac{1}{p} \sum_{y_j \in \mathcal{Y}_j} \mathbf{K}_{y_j}(\mathbf{s})\mathbf{f} \tag{5}$$

where $y_j \in \mathcal{Y}_j$ is a path from a set of p random walks starting from the j-th node.

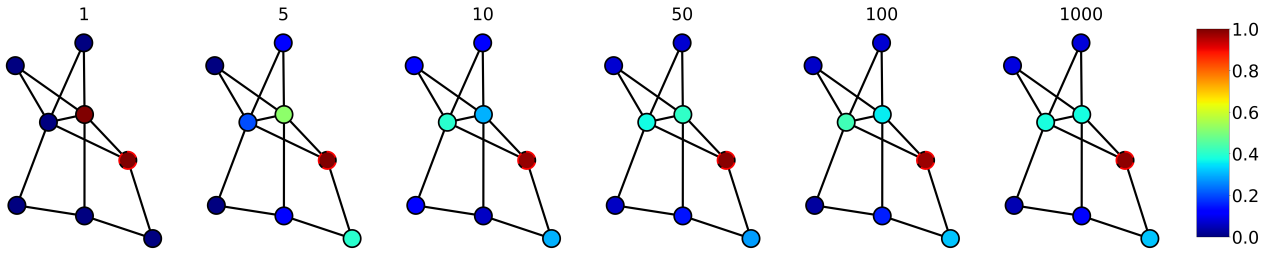

Figure 2. The effective spatial operator from a walk of length 3 starting at the dashed node, ranging from 1 to 1000 random walks.

3.3. Constructing pathGCN

So far we defined the path convolution, which operates in the spatial domain. In what follows, we omit $\mathbf{s}$ for brevity, and denote by $\mathbf{K}_{\mathcal{Y}}$ the pathConv module described in Eq. (5) on all the nodes in $\mathcal{V}$ given their corresponding path realizations $\mathcal{Y}$. To obtain a complete network, we need to add the channel mixing convolution and the non-linear activation σ, as follows

$$\mathbf{f}^{(l+1)} = \sigma(\mathbf{W}^{(l)}\mathbf{K}_{\mathcal{Y}}^{(l)}\mathbf{f}^{(l)}), \tag{6}$$

where in our implementation σ is ReLU and $\mathbf{W}^{(l)}$ is a 1×1 (i.e., point-wise) convolution . The simplest utilization of pathConv applies a single spatial operator parameterized by $\mathbf{s} \in \mathbb{R}^k$, shared among all channels and all L layers. While this is more flexible than using $\tilde{\mathbf{P}}$ as in Eq. (1), it can be further generalized. Scaling up, it is also possible to learn a different parameterization of the spatial operator per layer, that is, $\mathbf{s} \in \mathbb{R}^{L\times k}$. In practice and inspired by modern CNNs (Sandler et al., 2018; Ephrath et al., 2020), we found that learning a depth-wise spatial operator (i.e., a spatial operator per channel and layer such that $\mathbf{s} \in \mathbb{R}^{L\times c\times k}$) followed by a 1×1 convolution leads to favorable performance both accuracy- and computationally-wise across all the considered data sets in this paper, as reflected in our experiments in Sec. 4. We refer to this architecture as *pathGCN*. We note that it is also possible to learn different spatial operators for all pairs of channels, per layer – similarly to a standard fully-connected CNN.

Finally, learning the spatial convolution allows for a variety of operators, from non-smoothing (e.g., edge-detection filters) to smoothing operators. Thus, as we demonstrate in the experiments section, our network achieves state-of-the-art performance, and does not suffer from over-smoothing.

3.4. Convergence analysis

The process defined in Eq. (4)-(5) represents a simple stochastic process to build a spatial operation for a single channel that depends on two quantities. First, it depends on the algorithm used to sample the set of paths $\mathcal{Y}$, and second, it depends on the learnt weights $\mathbf{s}$. A natural question that arises is – does this process converge, and to what? To answer this question we need to make a mild assumption on the random walk process. Namely, we assume that the process is Markovian, that is, at each state the walk can visit any neighbouring node of the current node at an equal probability. In the following, we show that at the sampling limit (i.e., $p \to \infty$), there exists a stationary distribution of the paths, which induces a spatial operator sampled by our method, as exemplified in Fig. 2.

Let us consider the simple case of a convolution on a general path of length 2, which can be written as

$$\mathbf{K}_{y_j}(\mathbf{s})\mathbf{f} = s_0 f_{j_0} + s_1 f_{j_1} = s_0 f_j + s_1 f_{j_1}. \tag{7}$$

The second transition follows from the fact that the path originates from the j-th node, i.e., $f_{j_0} = f_j$. Note, that j_1 can represent any of the immediate neighbours of the j-th node. Therefore, the expectation of (7) which corresponds to all the nodes $\mathcal{V}$ is given by

$$\mathbb{E}_y(\mathbf{K}_y(\mathbf{s})\mathbf{f}) = s_0\mathbf{f} + s_1(\mathbf{A}\mathbf{D}^{-1})\mathbf{f} \tag{8}$$

where $\mathbf{A}$ is the adjacency matrix and $\mathbf{D}$ is a diagonal matrix with the degree of each node.

More generally, let us consider a path of an arbitrary length k. In this case we have that the expectation term reads

$$\mathbb{E}_y\left(\mathbf{K}_y(\mathbf{s})\mathbf{f}\right) = \left(\sum_{i=0}^{k-1} s_i(\mathbf{A}\mathbf{D}^{-1})^i\right)\mathbf{f} \tag{9}$$

where $(\mathbf{A}\mathbf{D}^{-1})^0 = \mathbf{I}$. We note that the transition matrix $\mathbf{A}\mathbf{D}^{-1}$ is column-stochastic, and therefore its eigenvalues are bounded between $[-1, 1]$. Thus, this polynomial formulation is stable and does not diverge for an appropriate choice of coefficients $\mathbf{s}$.

Furthermore, Eq. (9) is a deterministic representation of the process given in Eq. (5). If the number of path-realizations, p is large then the results from both are similar. For short paths, a direct evaluation of Eq. (9) can be computationally advantageous compared to its stochastic implementation. However, for long paths, computing the powers of the adjacency matrix times a vector can be expensive. The average

over paths is, in this case, an economical way to approximate the process, avoiding repeated matrix multiplications.

Lastly, as we show in our experiments, the stochastic nature of the process has additional advantages. In particular, it allows for better trainability and generalization, compared to the deterministic form. Indeed, a significant increase in the value of p causes performance degradation, as reported in Fig. 5. For inference purposes, we may use both the stochastic or deterministic formulations from Eq. (5) and (9) respectively, yielding similar results for both.

3.5. Computational cost and number of parameters

We compare the cost of our pathGCN with the methods considered in Eq. (1)-(3). On the spatial side, our pathGCN involves $n \times k \times p$ operations rather than $n \times d$ for the considered methods, where d is the mean node degree of the graph. Therefore, if $k \times p$ is larger than d, our method requires more computations. Nonetheless, our pathGCN has a wider aperture as it considers paths of k nodes, thus its field of view is larger . We also compare of the number of trainable parameters, which further highlights the difference of our approach. Recall that our pathGCN learns the spatial operator in addition to a 1×1 convolution, which is also present in other methods. That is, per pathGCN layer, the spatial weights that parameterize $\mathbf{K}_{\mathcal{Y}}$ require $c \times k$ parameters, while the 1×1 convolution $\mathbf{W}$ requires $c \times c$ parameters, where typically k is significantly smaller than c. In our experiments, different values of k and p were examined, and we found that setting k between 3 to 7 and p between 5 to 10 achieves better or on par with state-of-the-art models while keeping the computational cost reasonable.

We also consider the stochastic implementation of a random walk over an arbitrary graph. Sampling a single path of length k for all nodes in $\mathcal{V}$ requires $n \times k$ operations[1]. Therefore, if the number of paths p is smaller than the average node degree d of the graph, then the cost of random walk sampling is smaller than the cost of applying the adjacency matrix as in (9). Furthermore, in the deterministic case, a spatial operation on a path of length 2 requires $n \times d$ computations. Extending it to a path of length k requires $n \times d \times (k-1)$ operations, realized by polynomials of degree $k-1$ of the adjacency matrix. We note that Eq. (9), which is the deterministic form of our method, can also be used for inference purposes.

4. Experiments

We demonstrate our pathGCN on node classification and protein-protein interaction (Hamilton et al., 2017), followed by an ablation study in order to gain a profound understanding of our method. In all experiments, we employ a network that is comprised of an embedding layer (1×1 convolution), followed by a sequence of pathGCN layers, whose final output is fed to a 1×1 convolution layer which acts as a classifier. A detailed description of the network architecture is given in Appendix A. We use the Adam (Kingma & Ba, 2014) optimizer in all experiments, and perform grid search over the hyper-parameters of our network. The selected hyper-parameters are reported in Appendix B. The objective function in all experiments is the cross-entropy loss, besides inductive learning on PPI where we use the binary cross-entropy loss. Our code is implemented using PyTorch (Paszke et al., 2019) and PyTorch-Geometric (Fey & Lenssen, 2019) and trained on an Nvidia Titan RTX GPU.

We show that for all the considered tasks and datasets, whose statistics are provided in Tab. 1, our method is either better or on par with other state-of-the-art models.

[1] We note that further efficiency can be gained by using a tree structure that describes the nodes of the sampled paths.

Table 1. Node classification datasets statistics.

Dataset	Classes	Nodes	Edges	Features
Cora	7	2,708	5,429	1,433
Citeseer	6	3,327	4,732	3,703
Pubmed	3	19,717	44,338	500
Chameleon	5	2,277	36,101	2,325
Cornell	5	183	295	1,703
Texas	5	183	309	1,703
Wisconsin	5	251	499	1,703
PPI	121	56,944	818,716	50
Wiki-CS	10	11,701	216,123	300
Actor	5	7,600	33,544	932
Ogbn-arxiv	40	169,343	1,166,243	128

Table 2. Summary of semi-supervised node classification accuracy (%)

Method	Cora	Citeseer	Pubmed
ChebNet	81.2	69.8	74.4
GCN	81.1	70.8	79.0
GAT	83.1	70.8	78.5
APPNP	83.3	71.8	80.1
JKNET	81.1	69.8	78.1
GCNII	85.5	73.4	80.3
GRAND	84.7	73.6	81.0
PDE-GCN	84.3	75.6	80.6
EGNN	85.7	–	80.1
pathGCN (Ours)	**85.8**	**75.8**	**82.7**

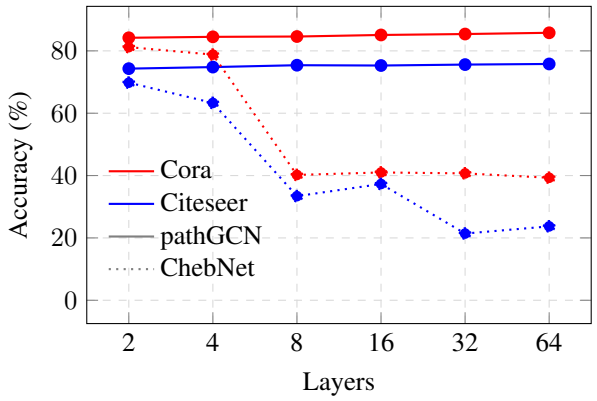

Figure 3. ChebNet vs pathGCN on semi-supervised node classification.

4.1. Semi-supervised node classification

Here, we use three datasets – Cora, Citeseer and Pubmed (Sen et al., 2008). For all datasets we use the standard training/validation/testing split as in (Yang et al., 2016), with 20 nodes per class for training, 500 validation nodes and 1,000 testing nodes and follow the training scheme of (Chen et al., 2020). For comparison, we consider various models like ChebNet (Defferrard et al., 2016), GCN (Kipf & Welling, 2016), GAT(Veličković et al., 2018), Inception (Szegedy et al., 2017), APPNP (Klicpera et al., 2019), JKNet (Xu et al., 2018), DropEdge (Rong et al., 2020), GCNII(Chen et al., 2020), GRAND (Chamberlain et al., 2021), PDE-GCN (Eliasof et al., 2021) and EGNN (Zhou et al., 2021).

As discussed in Sec. 3, our pathGCN is constructed such that wider spatial operators are obtained, allowing for improved expressiveness, namely, compared to methods that are based on the graph Laplacian or its proxies. As objectively portrayed by Tab. 2 and 3, our pathGCN is capable of obtaining higher accuracy on all three datasets. A bold improvement is obtained on Pubmed, where an accuracy of 82.7% is achieved, compared to previously state-of-the-art $\text{GRAND}_{\text{nl-rw}}$ with 81.0%. Our approach also benefits from the inherent absence of over-smoothing, as the spatial operator is fully learnt. This is also validated by inspecting the learnt kernels of different layers of the network. Indeed, as shown in Fig. 4, some layers perform smoothing by averaging, while others act as edge-detection filters by considering the difference of neighbouring nodes along the path.

4.2. Fully-supervised node classification

To further validate our method, we employ a total of 10 datasets. First, we follow (Pei et al., 2020) and examine our pathGCN on Cora, Citeseer, Pubmed, Chameleon (Rozemberczki et al., 2021), Cornell, Texas and Wisconsin. We also use the same train/validation/test splits of $60\%, 20\%, 20\%$, respectively, and report the average performance over 10 random splits from (Pei et al., 2020). We fix the number of channels to 64 and perform grid search to determine the hyper-parameters. We compare our network with GCN, GAT, Geom-GCN (Pei et al., 2020), APPNP, JKNet, Inception, GCNII and PDE-GCN in Tab. 4. Our experiments read improvement across all data-sets compared to all the considered methods. For instance, we obtain 90.02% accuracy on Cora with our pathGCN, compared to 88.49% of GCNII* and 88.60% of PDE-GCN. In addition, we examine our pathGCN on larger datasets using the standard train/validation/test splits of Actor (Pei et al., 2020), Ogbn-arxiv (Hu et al., 2020) and Wiki-CS (20 random splits) (Mernyei & Cangea, 2020) in Tab. 5 and 6 – where again we see accuracy improvement across all considered datasets.

Table 3. Semi-supervised node classification accuracy (%). – indicates not available results.

Dataset	Method	Layers 2	4	8	16	32	64
Cora	GCN	**81.1**	80.4	69.5	64.9	60.3	28.7
	GCN (Drop)	**82.8**	82.0	75.8	75.7	62.5	49.5
	JKNet	–	80.2	80.7	80.2	**81.1**	71.5
	JKNet (Drop)	–	**83.3**	82.6	83.0	82.5	83.2
	Incep	–	77.6	76.5	81.7	**81.7**	80.0
	Incep (Drop)	–	82.9	82.5	83.1	83.1	**83.5**
	GCNII	82.2	82.6	84.2	84.6	85.4	**85.5**
	GCNII*	80.2	82.3	82.8	83.5	84.9	**85.3**
	$\text{PDE-GCN}_{\text{D}}$	82.0	83.6	84.0	84.2	84.3	**84.3**
	EGNN	83.2	–	–	85.4	–	**85.7**
	pathGCN (Ours)	84.2	84.5	84.6	85.1	85.4	**85.8**
Citeseer	GCN	**70.8**	67.6	30.2	18.3	25.0	20.0
	GCN (Drop)	**72.3**	70.6	61.4	57.2	41.6	34.4
	JKNet	–	68.7	67.7	**69.8**	68.2	63.4
	JKNet (Drop)	–	72.6	71.8	**72.6**	70.8	72.2
	Incep	–	69.3	68.4	**70.2**	68.0	67.5
	Incep (Drop)	–	**72.7**	71.4	72.5	72.6	71.0
	GCNII	68.2	68.8	70.6	72.9	**73.4**	73.4
	GCNII*	66.1	66.7	70.6	72.0	**73.2**	73.1
	$\text{PDE-GCN}_{\text{D}}$	74.6	75.0	75.2	75.5	**75.6**	75.5
	pathGCN (Ours)	74.3	74.8	75.4	75.3	75.6	**75.8**
Pubmed	GCN	**79.0**	76.5	61.2	40.9	22.4	35.3
	GCN (Drop)	**79.6**	79.4	78.1	78.5	77.0	61.5
	JKNet	–	78.0	**78.1**	72.6	72.4	74.5
	JKNet (Drop)	–	78.7	78.7	**79.7**	79.2	78.9
	Incep	–	77.7	**77.9**	74.9	–	–
	Incep (Drop)	–	**79.5**	78.6	79.0	–	–
	GCNII	78.2	78.8	79.3	**80.2**	79.8	79.7
	GCNII*	77.7	78.2	78.8	**80.3**	79.8	80.1
	$\text{PDE-GCN}_{\text{D}}$	79.3	**80.6**	80.1	80.4	80.2	80.3
	EGNN	79.2	–	–	80.0	–	**80.1**
	pathGCN (Ours)	81.8	81.8	82.4	82.5	82.4	**82.7**

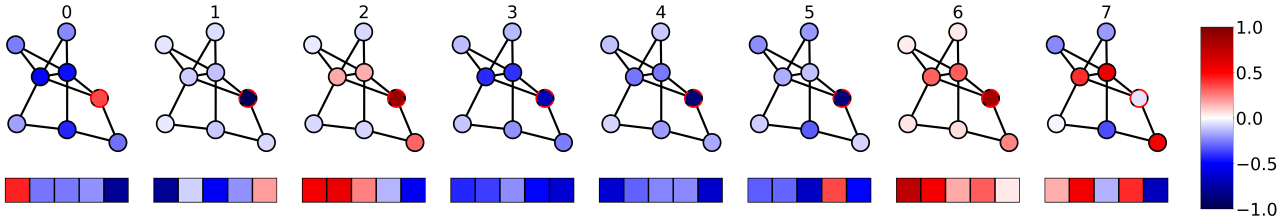

Figure 4. A visualization of the learnt spatial operator $\mathbf{K}_y$ from a 8-layer pathGCN trained on Cora, applied on the dashed node in the graph (top), and its corresponding learnt weights (bottom).

Table 4. Fully-supervised node classification accuracy (%). (L) denotes the number of layers.

Method	Cora	Cite.	Pubm.	Cham.	Corn.	Texas	Wisc.
GCN (Kipf & Welling, 2016)	85.77	73.68	88.13	28.18	52.70	52.16	45.88
GAT (Veličković et al., 2018)	86.37	74.32	87.62	42.93	54.32	58.38	49.41
Geom-GCN-I (Pei et al., 2020)	85.19	77.99	90.05	60.31	56.76	57.58	58.24
Geom-GCN-P (Pei et al., 2020)	84.93	75.14	88.09	60.90	60.81	67.57	64.12
Geom-GCN-S (Pei et al., 2020)	85.27	74.71	84.75	59.96	55.68	59.73	56.67
APPNP (Klicpera et al., 2019)	87.87	76.53	89.40	54.30	73.51	65.41	69.02
JKNet (Xu et al., 2018)	85.25 (16)	75.85 (8)	88.94 (64)	60.07 (32)	57.30 (4)	56.49 (32)	48.82 (8)
JKNet (Drop) (Rong et al., 2020)	87.46 (16)	75.96 (8)	89.45 (64)	62.08 (32)	61.08 (4)	57.30 (32)	50.59 (8)
Incep (Drop) (Rong et al., 2020)	86.86 (8)	76.83 (8)	89.18 (4)	61.71 (8)	61.62 (16)	57.84 (8)	50.20 (8)
GCNII (Chen et al., 2020)	88.49 (64)	77.08 (64)	89.57 (64)	60.61 (8)	74.86 (16)	69.46 (32)	74.12 (16)
GCNII* (Chen et al., 2020)	88.01 (64)	77.13 (64)	90.30 (64)	62.48 (8)	76.49 (16)	77.84 (32)	81.57 (16)
PDE-GCN$_{\text{M}}$(Eliasof et al., 2021)	88.60 (16)	78.48 (32)	89.93 (16)	66.01 (16)	89.73 (64)	93.24 (32)	91.76 (16)
pathGCN (Ours)	**90.02** (64)	**78.95** (32)	**90.42** (64)	**66.79** (16)	**91.35** (8)	**95.14** (16)	**93.53** (16)

Table 5. Fully-supervised node classification accuracy (%).

Method	Actor	Ogbn-arxiv
GCN (Kipf & Welling, 2016)	26.86	71.74
GAT (Veličković et al., 2018)	28.45	71.89
APPNP (Klicpera et al., 2019)	31.26	71.82
Geom-GCN-P (Pei et al., 2020)	31.63	–
JKNet (Xu et al., 2018)	29.81	72.19
SGC (Wu et al., 2019)	30.98	69.20
GCNII (Chen et al., 2020)	32.87	72.74
EGNN (Zhou et al., 2021)	–	72.70
GRAND (Chamberlain et al., 2021)	–	72.23
pathGCN (Ours)	**37.54**	**72.83**

Table 6. Node classification on Wiki-CS.

Method	Accuracy (%)
GCN (Kipf & Welling, 2016)	77.19
GAT (Veličković et al., 2018)	77.65
SuperGAT (Kim & Oh, 2020)	77.90
APPNP (Klicpera et al., 2019)	79.84
pathGCN (Ours)	**80.02**

4.3. Inductive learning

We employ the PPI dataset (Hamilton et al., 2017) for the inductive learning task. We use a 8 layer pathGCN, without weight-decay, dropout of 0.2 and a learning rate of 0.001. We compare our results with various methods like GraphSAGE, GAT, JKNet, GeniePath, Cluster-GCN, GCNII and others, and present the micro-averaged F1 score in in Tab. 7. We note that our pathGCN achieves a score of 99.61, superior to methods like GAT, JKNet, GeniePath, PDE-GCN, and slightly above GCNII* with a score of 99.58.

4.4. Graph classification

Our previous experiments considered various datasets and settings of the node classification task. To further demonstrate the efficacy of our pathGCN we experiment with graph classification on the popular TUDatasets (Morris et al., 2020). We follow the same experimental settings from (Xu et al., 2019) and report the 10 fold cross-validation performance on MUTAG, PTC, PROTEINS and NCI1 datasets. The hyper-parameters are determined by grid search, as

Table 7. Inductive learning on PPI dataset. Results are reported in micro-averaged F1 score.

Method	Micro-averaged F1
GraphSAGE (Hamilton et al., 2017)	61.20
VR-GCN (Chen et al., 2018)	97.80
GaAN (Zhang et al., 2018a)	98.71
GAT (Veličković et al., 2018)	97.30
JKNet (Xu et al., 2018)	97.60
GeniePath (Liu et al., 2018)	98.50
Cluster-GCN (Chiang et al., 2019)	99.36
GCNII (Chen et al., 2020)	99.54
GCNII* (Chen et al., 2020)	99.58
PDE-GCN$_{\mathrm{M}}$ (Eliasof et al., 2021)	99.18
pathGCN (ours)	**99.61**

Table 8. TUDatasets graph classification accuracy (%).

Model	MUTAG	PTC	PROTEINS	NCI1
DGCNN	$85.8_{\pm 1.8}$	$58.6_{\pm 2.5}$	$75.5_{\pm 0.9}$	$74.4_{\pm 0.5}$
IGN	$83.9_{\pm 13.0}$	$58.5_{\pm 6.9}$	$76.6_{\pm 5.5}$	$74.3_{\pm 2.7}$
GIN	$89.4_{\pm 5.6}$	$64.6_{\pm 7.0}$	$76.2_{\pm 2.8}$	$82.7_{\pm 1.7}$
CIN	$92.7_{\pm 6.1}$	$68.2_{\pm 5.6}$	$77.0_{\pm 4.3}$	$\mathbf{83.6_{\pm 1.4}}$
GSN	$92.7_{\pm 7.5}$	$68.2_{\pm 7.2}$	$76.6_{\pm 5.0}$	$83.5_{\pm 2.0}$
pathGCN (ours)	$\mathbf{94.7_{\pm 4.7}}$	$\mathbf{75.2_{\pm 5.3}}$	$\mathbf{80.4_{\pm 4.2}}$	$83.3_{\pm 1.3}$

in (Xu et al., 2019) and are reported in Appendix B. We compare our pathGCN with recent methods like DGCNN (Zhang et al., 2018b), IGN (Maron et al., 2018),GIN (Xu et al., 2019), CIN (Bodnar et al., 2021) and GSN (Bouritsas et al., 2022). The results are summarized in Tab. 8, where our pathGCN shows better or similar results compared to the considered methods, further highlighting the efficacy of our approach.

4.5. Inference and runtimes

In this section we compare the inference accuracy of our stochastic approach with its deterministic form. Later, we report the runtimes of our method.

As discussed in Sec. 3.4, both the stochastic and deterministic forms of our pathGCN can be used during inference. In Tab. 9, we show that after training our stochastic pathGCN, it is possible to obtain similar results (with up to 0.1% accuracy difference) by its deterministic form as in Eq. (9). The stochastic results are averaged over 10 inference runs, and we also present their standard deviation.

Next, we present the training and inferences times, as well as path sampling time of our pathGCN and compare it with

Table 9. Deterministic vs stochastic pathGCN inference accuracy (%).

Inference	Cora	Cite.	Pub.	ogbn-arxiv	Wisc.	PPI
Determin.	85.8	75.8	82.7	72.84	93.51	99.61
Stochastic	85.8 ± 0.29	75.8 ± 0.34	82.7 ± 0.34	72.83 ± 0.13	93.53 ± 0.21	99.61 ± 0.02

Table 10. Computation times (in ms) on Cora.

Model	Path samp.	Training	Inference	Acc (%)
GCN (2 layers)	-	4.07	1.97	81.1
GCNII (2 layers)	-	4.24	1.95	82.2
GCNII (8 layers)	-	13.05	6.87	84.2
GAT (2 layers)	-	5.27	1.96	83.1
pathGCN$_{k=2,p=5}$ (2 layers)	0.378	3.48	1.67	81.3
pathGCN$_{k=5,p=5}$ (2 layers)	0.384	5.35	2.17	84.2

GCN, GCNII and GAT, in Tab. 10. For reference, we also report the node classification accuracy of the different methods. We see that the sampling time is relatively small, and that our pathGCN has a similar runtime to GCN, GCNII and GAT, while obtaining similar or better accuracy. For example, our pathGCN with $k = 5, p = 5$ obtains an accuracy of 84.2% on Cora and requires 5.35 milliseconds (ms) for a training iteration, while GCNII requires 8 layers and 13.05 ms to achieve the same accuracy.

4.6. Ablation study

In this section we consider the different possible variants of our pathGCN, and the influence of our its hyper-parameters the path length k and the number of paths p.

As discussed in Sec. 3, our method can be formulated in various manners. That is, it is possible to learn a *global* spatial operator, which is most similar to GCN (Kipf & Welling, 2016), only there it is fixed and not learnt. We denote this variant by pathGCN$_{\mathrm{G}}$. In addition, we can learn a spatial operator *per layer* that is shared among the channels, denoted by pathGCN$_{\mathrm{PL}}$. The next step is our pathGCN from Sec. 3.3, which is more similar to CNNs, by utilizing a *depth-wise* spatial operator (i.e., per channel and layer). As depicted in Tab. 11, the global variant, pathGCN$_{\mathrm{G}}$ yields the least attractive performance among the three considered variants. This is not surprising, as a global operator is learnt, which is less expressive than pathGCN$_{\mathrm{PL}}$ and pathGCN. Still, it is interesting to see that no over-smoothing is evident. As discussed in (Wu et al., 2019), a recurrent application of a *fixed* operator leads to over-smoothing. However, the learnt spatial operator of pathGCN$_{\mathrm{G}}$ is variable. Following that, the per layer variant, pathGCN$_{\mathrm{PL}}$ prevents over-smoothing and also further improves accuracy. This is obtainable as the network has the freedom to learn a variety of kernels, which may function as smoothing kernels or edge detectors.

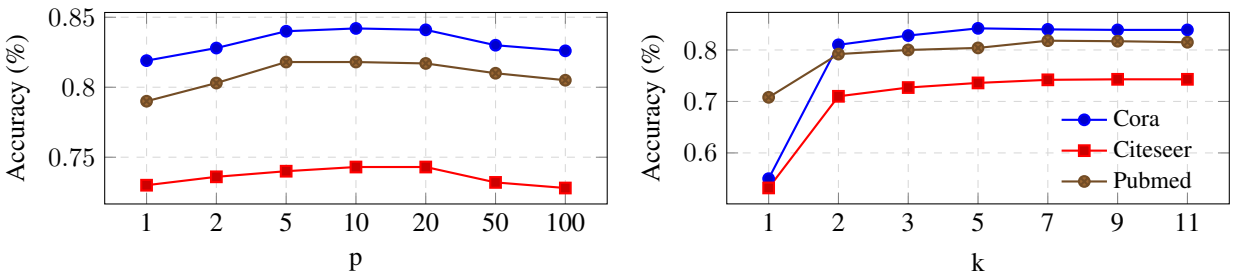

Figure 5. Semi-supervised node classification accuracy (%), as a function of number of paths p and path length k.

Table 11. Variants of pathGCN on semi-supervised classification. Results are reported in accuracy (%).

Dataset	Method	Layers 2	4	8	16	32	64
Cora	pathGCN$_G$	83.0	81.9	81.5	81.2	81.4	82.0
	pathGCN$_{PL}$	82.9	83.3	83.5	83.8	84.1	84.7
	pathGCN	84.2	84.5	84.6	85.1	85.4	85.8
Citeseer	pathGCN$_G$	73.1	71.9	72.0	71.9	72.6	71.7
	pathGCN$_{PL}$	73.4	73.6	74.0	74.3	74.5	75.0
	pathGCN	74.3	74.8	75.4	75.3	75.6	75.8
Pubmed	pathGCN$_G$	80.9	80.4	81.1	81.0	80.4	80.8
	pathGCN$_{PL}$	81.1	81.2	81.5	82.0	82.2	82.1
	pathGCN	81.8	81.8	82.4	82.5	82.4	82.7

Finally, we see that the additional degrees of freedom in the depth-wise operator in pathGCN lead to overall better performance, also as depicted in our other experiments.

As for the hyper-parameters of our method, we report in Fig. 5 the influence of the number of paths p and path length k on the performance of our pathGCN on semi-supervised node classification using Cora, Citeseer and Pubmed. In all the experiments we use a 2 layer pathGCN, fix one hyper-parameter, and vary the other. Specifically, for the evaluation of the influence of number of paths p, we fix the path length to $k = 5$, and the results in Fig. 5 suggest that indeed the stochastic nature of our method is beneficial to obtain higher accuracy, since we see a consistent accuracy degradation trend as k is increased past 20. In addition, we present in Fig. 5, that by fixing $p = 10$ and examining a variable path length k from 1 to 11, accuracy improves and stagnates at $k = 7$. We can see that when $k = 1$, our pathGCN behaves similarly to an MLP (Qi et al., 2017) (as it considers only the self node), and that in some cases, increasing the kernel size caused a slight performance degradation.

5. Conclusion

In this paper we propose a new approach for learning the spatial operators for GCNs. Our motivation stems from the need for deep GCNs that have expressive spatial kernels, similar to standard CNNs that do not over-smooth. Our approach leverages on paths defined on the graph, to enable the learning of such operators, further bridging the gap between GCNs and CNNs.

Just as the Laplacian is not the sole spatial operator used on images in CNNs, it may also not necessarily be optimal in the case of graphs and GCNs. To this end we propose *pathGCN* which replaces the Laplacian based operator by a fully learnt kernel. Indeed, our experiments reveal that more expressive kernels can be learnt based on the data and task at hand, leading to consistently better accuracy on numerous applications and datasets and without over-smoothing.

Acknowledgements

The research reported in this paper was supported by the Israel Innovation Authority through Avatar consortium. In addition, this work was supported in part by the Israeli Council for Higher Education (CHE) via the Data Science Research Center, Ben-Gurion University of the Negev, Israel. ME is supported by Kreitman High-tech scholarship.

References

Bodnar, C., Frasca, F., Otter, N., Wang, Y., Lio, P., Montufar, G. F., and Bronstein, M. Weisfeiler and lehman go cellular: Cw networks. *Advances in Neural Information Processing Systems*, 34:2625–2640, 2021.

Boscaini, D., Masci, J., Rodolà, E., and Bronstein, M. Learning shape correspondence with anisotropic convolutional neural networks. 05 2016.

Bouritsas, G., Frasca, F., Zafeiriou, S. P., and Bronstein, M. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4): 18–42, 2017.

Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

Chamberlain, B. P., Rowbottom, J., Gorinova, M., Webb, S., Rossi, E., and Bronstein, M. M. Grand: Graph neural diffusion. *arXiv preprint arXiv:2106.10934*, 2021.

Chen, J., Zhu, J., and Song, L. Stochastic training of graph convolutional networks with variance reduction. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 942–950. PMLR, 10–15 Jul 2018. URL http://proceedings.mlr.press/v80/chen18p.html.

Chen, M., Wei, Z., Huang, Z., Ding, B., and Li, Y. Simple and deep graph convolutional networks. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 1725–1735. PMLR, 13–18 Jul 2020. URL http://proceedings.mlr.press/v119/chen20v.html.

Chiang, W.-L., Liu, X., Si, S., Li, Y., Bengio, S., and Hsieh, C.-J. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2019. URL https://arxiv.org/pdf/1905.07953.pdf.

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

Eliasof, M. and Treister, E. Diffgcn: Graph convolutional networks via differential operators and algebraic multigrid pooling. *34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.*, 2020.

Eliasof, M., Haber, E., and Treister, E. Pde-gcn: Novel architectures for graph neural networks motivated by partial differential equations. *Advances in Neural Information Processing Systems*, 34, 2021.

Ephrath, J., Eliasof, M., Ruthotto, L., Haber, E., and Treister, E. Leanconvnets: Low-cost yet effective convolutional neural networks. *IEEE Journal of Selected Topics in Signal Processing*, 2020.

Fey, M. and Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Gasteiger, J., Weißenberger, S., and Günnemann, S. Diffusion improves graph learning. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

Grover, A. and Leskovec, J. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864, 2016.

Hamilton, W. L., Ying, R., and Leskovec, J. Inductive representation learning on large graphs. In *NIPS*, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

Kim, D. and Oh, A. How to find your friendly neighborhood: Graph attention design with self-supervision. In *International Conference on Learning Representations*, 2020.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Klicpera, J., Bojchevski, A., and Günnemann, S. Combining neural networks with personalized pagerank for classification on graphs. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=H1gL-2A9Ym.

Krizhevsky, A., Sutskever, I., and Hinton, G. Imagenet classification with deep convolutional neural networks. *Adv Neural Inf Process Syst*, 61:1097–1105, 2012.

Lei, X. and Bian, C. Integrating random walk with restart and k-nearest neighbor to identify novel circrna-disease association. *Scientific reports*, 10(1):1–9, 2020.

Levie, R., Monti, F., Bresson, X., and Bronstein, M. M. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Transactions on Signal Processing*, 67(1):97–109, 2018.

Liu, Z., Chen, C., Li, L., Zhou, J., Li, X., and Song, L. Geniepath: Graph neural networks with adaptive receptive paths. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33, 02 2018. doi: 10.1609/aaai.v33i01.33014424.

Ma, Z., Xuan, J., Wang, Y. G., Li, M., and Liò, P. Path integral based convolution and pooling for graph neural networks. *arXiv preprint arXiv:2006.16811*, 2020.

Maron, H., Ben-Hamu, H., Shamir, N., and Lipman, Y. Invariant and equivariant graph networks. *ICLR*, 2018.

Mernyei, P. and Cangea, C. Wiki-cs: A wikipedia-based benchmark for graph neural networks. *arXiv preprint arXiv:2007.02901*, 2020.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pp. 3111–3119, 2013.

Monti, F., Boscaini, D., Masci, J., Rodola, E., Svoboda, J., and Bronstein, M. M. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5115–5124, 2017.

Morris, C., Kriege, N. M., Bause, F., Kersting, K., Mutzel, P., and Neumann, M. Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020. URL www.graphlearning.io.

Nikolentzos, G. and Vazirgiannis, M. Random walk graph neural networks. *Advances in Neural Information Processing Systems*, 33:16211–16222, 2020.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Pei, H., Wei, B., Chang, K. C.-C., Lei, Y., and Yang, B. Geom-gcn: Geometric graph convolutional networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1e2agrFvS.

Perozzi, B., Al-Rfou, R., and Skiena, S. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 701–710, 2014.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 652–660, 2017.

Rong, Y., Huang, W., Xu, T., and Huang, J. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=Hkx1qkrKPr.

Rozemberczki, B., Allen, C., and Sarkar, R. Multi-Scale Attributed Node Embedding. *Journal of Complex Networks*, 9(2), 2021.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

Strokach, A., Becerra, D., Corbi-Verge, C., Perez-Riba, A., and Kim, P. M. Fast and flexible protein design using deep graph neural networks. *Cell Systems*, 11(4):402 – 411.e4, 2020. ISSN 2405-4712. doi: https://doi.org/10.1016/j.cels.2020.08.016. URL http://www.sciencedirect.com/science/article/pii/S2405471220303276.

Sun, X., Rosin, P. L., Martin, R. R., and Langbein, F. C. Random walks for feature-preserving mesh denoising. *Computer Aided Geometric Design*, 25(7):437–456, 2008.

Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. A. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph Attention Networks. *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M., and Solomon, J. M. Dynamic graph cnn for learning on point clouds. *arXiv preprint arXiv:1801.07829*, 2018.

Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. Simplifying graph convolutional networks. In *International conference on machine learning*, pp. 6861–6871. PMLR, 2019.

Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.-i., and Jegelka, S. Representation learning on graphs with jumping knowledge networks. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5453–5462. PMLR, 10–15 Jul 2018. URL http://proceedings.mlr.press/v80/xu18c.html.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ryGs6iA5Km.

Yang, Z., Cohen, W., and Salakhudinov, R. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pp. 40–48. PMLR, 2016.

Zhang, J., Shi, X., Xie, J., Ma, H., King, I., and Yeung, D. Gaan: Gated attention networks for learning on large and spatiotemporal graphs. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence*, pp. 339–349, 2018a.

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

Zhao, L. and Akoglu, L. Pairnorm: Tackling oversmoothing in gnns. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rkecl1rtwB.

Zhou, K., Huang, X., Zha, D., Chen, R., Li, L., Choi, S.-H., and Hu, X. Dirichlet energy constrained learning for deep graph neural networks. *Advances in Neural Information Processing Systems*, 34, 2021.

A. Architecture in details

We elaborate on the architecture that was used in our experiments. As discussed in Sec. 4, our network is comprised of an embedding layer (1×1 convolution), a sequence of pathGCN layers, and a closing (projection) layer (1×1 convolution). Throughout this section, c_{in} and c_{out} denote the input and output channels, respectively, and c denotes the number of features in hidden layers (which is a hyper-parameter, as given in Appendix B) We denote the number of pathGCN blocks by L, and the dropout probability by p_{drop}. Our architecture for node classification tasks is described in Tab. 12, which is similar to the architecture found in GCNII (Chen et al., 2020), only with our pathGCN instead. In Tab. 13 we present our network for graph classification tasks, which is based on the one in GIN (Xu et al., 2019).

Table 12. pathGCN architecture for node classification.

Input size	Layer	Output size
$n \times c_{in}$	1×1 Dropout(p_{drop})	$n \times c_{in}$
$n \times c_{in}$	1×1 Convolution	$n \times c$
$n \times c$	ReLU	$n \times c$
$n \times c$	$L\times$ pathGCN block	$n \times c$
$n \times c$	1×1 Dropout(p_{drop})	$n \times c$
$n \times c$	1×1 Convolution	$n \times c_{out}$

Table 13. pathGCN architecture for graph classification.

Input size	Layer	Output size
$n \times c_{in}$	1×1 Convolution	$n \times c$
$n \times c$	ReLU	$n \times c$
$n \times c$	$L \times$ [pathGCN , BN , 1×1 Convolution, ReLU]	$n \times c$
$n \times c$	1×1 Add-pool	$1 \times c$
$1 \times c$	1×1 Convolution	$1 \times c$
$1 \times c$	1×1 Dropout(p_{drop})	$1 \times c$
$1 \times c$	1×1 Convolution	$1 \times c_{out}$

B. Hyper-parameters details

We provide the selected hyper-parameters in our experiments. We denote the learning rate of our pathGCN layers by LR_{GCN}, and the learning rate of the 1×1 opening (embedding) and closing (classifier) layers by LR_{oc}. Also, the weight decay for the opening and closing layers is denoted by WD_{oc}, and for the pathGCN layers by WD_{GCN}.

Table 14. Semi-supervised node classification hyper-parameters

Dataset	LR_{GCN}	WD_{GCN}	LR_{oc}	WD_{oc}	c	p_{drop}	k	p
Cora	$1 \cdot 10^{-3}$	$2 \cdot 10^{-5}$	$1 \cdot 10^{-2}$	$1 \cdot 10^{-5}$	64	0.6	5	5
Citeseer	$1 \cdot 10^{-3}$	$1 \cdot 10^{-5}$	$7 \cdot 10^{-3}$	$5 \cdot 10^{-5}$	256	0.7	5	5
Pubmed	$5 \cdot 10^{-3}$	0	$1 \cdot 10^{-2}$	$1 \cdot 10^{-5}$	256	0.5	7	10

Table 15. Fully-supervised node classification hyper-parameters

Dataset	LR_{GCN}	WD_{GCN}	LR_{oc}	WD_{oc}	c	p_{drop}	k	p
Cora	$1 \cdot 10^{-4}$	$1 \cdot 10^{-4}$	$7 \cdot 10^{-2}$	$1 \cdot 10^{-4}$	64	0.5	5	10
Citeseer	$3 \cdot 10^{-4}$	$5 \cdot 10^{-5}$	$8 \cdot 10^{-3}$	$1 \cdot 10^{-4}$	64	0.5	5	10
Pubmed	$1 \cdot 10^{-4}$	$2 \cdot 10^{-4}$	$1 \cdot 10^{-2}$	$1 \cdot 10^{-6}$	64	0.5	7	10
Chameleon	$5 \cdot 10^{-4}$	$1 \cdot 10^{-5}$	$5 \cdot 10^{-3}$	$3 \cdot 10^{-5}$	64	0.5	3	10
Cornell	$4 \cdot 10^{-4}$	$1 \cdot 10^{-5}$	$5 \cdot 10^{-2}$	$5 \cdot 10^{-4}$	64	0.5	5	10
Texas	$3 \cdot 10^{-4}$	$5 \cdot 10^{-4}$	$4 \cdot 10^{-2}$	$1 \cdot 10^{-4}$	64	0.5	7	10
Wisconsin	$3 \cdot 10^{-4}$	$2 \cdot 10^{-4}$	$1 \cdot 10^{-2}$	$5 \cdot 10^{-5}$	64	0.5	7	10
Actor	$2 \cdot 10^{-4}$	$1 \cdot 10^{-4}$	$8 \cdot 10^{-2}$	$5 \cdot 10^{-4}$	64	0.5	7	10
Wiki-CS	$3 \cdot 10^{-2}$	$1 \cdot 10^{-4}$	$7 \cdot 10^{-3}$	$1 \cdot 10^{-5}$	64	0.3	7	5
Ogbn-arxiv	$1 \cdot 10^{-3}$	0	$1 \cdot 10^{-3}$	0	256	0.1	5	10

B.1. Semi-supervised node classification

The hyper-parameters for this experiment are summarized in Tab. 14.

B.2. Fully-supervised node classification

The hyper-parameters for this experiment are summarized in Tab. 15.

B.3. Inductive learning on PPI

For this experiment we used $LR_{oc} = LR_{GCN} = 0.001$, with dropout probability $p_{drop} = 0.2$, $k = 5$ and $p = 10$, and no weight decay was used.

B.4. Graph classification

The hyper-parameters in this experiment were chosen according to the protocol described in (Xu et al., 2019). We present the chosen hyper-parameters in Tab. 16. Throughout all the experiments in this section, no weight decay was used.

B.5. Ablation study

In this experiment we used the same hyper-parameters as reported in Tab. 14, for the results in Tab. 11. For the results in Fig. 5, we use the same learning rate and weight decay, but k and p are as described in the main paper.

Table 16. Graph classification hyper-parameters. BS denotes batch size.

Dataset	LR_{GCN}	LR_{oc}	c	p_{drop}	BS	p	k
MUTAG	0.01	0.01	32	0	32	5	5
PTC	0.01	0.01	32	0	32	5	5
PROTEINS	0.01	0.01	32	0	128	5	10
NCI1	0.01	0.01	32	0.5	32	5	10