PDE-GCN: Novel Architectures for Graph Neural Networks Motivated by Partial Differential Equations

Anonymous Author(s)
Affiliation
Address
email

Abstract

Graph neural networks are increasingly becoming the go-to approach in various fields such as computer vision, computational biology and chemistry, where data are naturally explained by graphs. However, unlike traditional convolutional neural networks, deep graph networks do not necessarily yield better performance than shallow graph networks. This behavior usually stems from the over-smoothing phenomenon. In this work, we propose a family of architectures to control this behavior by design. Our networks are motivated by numerical methods for solving Partial Differential Equations (PDEs) on manifolds, and as such, their behavior can be explained by similar analysis. Moreover, as we demonstrate using an extensive set of experiments, our PDE-motivated networks can generalize and be effective for various types of problems from different fields. Our architectures obtain better or on par with the current state-of-the-art results for problems that are typically approached using different architectures.

1 Introduction

In recent years, Graph Convolutional Networks (GCNs) [1, 2, 3] have drawn the attention of researchers and practitioners in a variety of domains and applications, ranging from computer vision and graphics [4, 5, 6, 7] to computational biology [8, 9, 10], recommendation systems [11] and social network analysis [12, 13]. However, GCNs still suffer from two main problems. First, they are usually *shallow* as opposed to the concept of deep convolutional neural networks (CNNs) [14, 15] due to the *over-smoothing* phenomenon [16, 17, 18], where the node feature vectors become almost identical, such that they are indistinguishable, which yields non-optimal performance. Furthermore, GCNs are typically customized to a specific domain and application. That is, as we demonstrate in Sec. 4.1, a successful point-cloud classification network [5] can perform poorly on a citation graph node-classification problem [18], and vice-versa. Furthermore, because many GCNs lack theoretical guarantees, it is difficult to reason about their success on one problem and lack on others. These observations motivate us to develop a profound understanding of graph networks and their dynamics.

To this end, we suggest a novel, universal approach to the design of GCN architectures. Our inspiration stems from the similarities and equivalence between Partial Differential Equations (PDEs) and deep networks explored in [19, 20, 21]. Furthermore, as GCNs can be thought of as a generalization of CNNs, and a standard convolution can be represented as a combination of differential operators on a structured grid [21], we adopt this interpretation to explore versions of GCNs as PDEs on graphs or manifolds. We therefore call our network architectures *PDE-GCN*, and demonstrate that our approach is general with respect to the given task. That is, our architectures behave similarly for different problems, and their performance is on par or better than other domain-specific GCNs. Furthermore, our family of architectures are backed by theoretical guarantees that allow us to explain the behavior of the GCNs in some of the results that we present. To be more specific, our contribution is as follows:

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

- We introduce and implement general graph convolution operators, based on graph gradient and divergence. This abstraction of the spatial operation on the graph leads to a more general and flexible approach for architecture design.
- We propose treating a variety of graph related problems as discretized PDEs, and formulate the dynamics that match different problems such as node-classification and dense shape-correspondence. This is in direct effort to propose a family of networks that can solve multiple problems, instead of GCNs which are tailored for a specific application.
- Our method allows constructing a deep GCN without over-smoothing, with theoretical guarantees.
- We validate our method by conducting numerous experiments on multiple datasets and applications, achieving significantly better or similar accuracy compared to state-of-the-art models.

2 Related work

Graph Convolutional Networks: GCNs are typically divided into spectral [1, 3, 2] and spatial [22, 23, 24, 4] categories. Most of those can be implemented using the Message-Passing Neural Network paradigm [24], where each node aggregates features (messages) from its neighbors, according to some scheme. The works [3, 2] use polynomials of the graph Laplacian to parameterize the convolution operator. DGCNN [5] constructs a k-nearest-neighbors graph from point-clouds and dynamically updates it. MoNet [4] learns a Gaussian mixture model to weight the edges of meshes for shape analysis tasks. Works like GraphSAGE [25] and GAT [26] propose methods for inductive and transductive learning on non-geometrical graphs.

Several of the methods above suffer from over-smoothing [16, 18], leading to undesired node features similarity for deep networks. To overcome this problem, some approaches rely on imposing regularization and augmentation. For example, PairNorm [16] introduces a novel normalization layer, and DropEdge [27] randomly removes edges to decrease the degree of nodes. Other methods prevent over-smoothing by dedicated construction. For instance, JKNet [28] combines all intermediate representations at each stage of the network. APPNP [29] replaces learnt convolutional layers with a pre-defined kernel based on PageRank, yielding a shallow network which preserves locality, making it robust to over-smoothing. GCNII [18] proposes to add an identity residual where the initial features of the networks are added to the features of the l-th layer, scaled by some coefficient.

Another approach is to construct a network that inherently does not over-smooth, as we suggest in this work. Our network is based on discretized PDEs, hence we are able to motivate our choices by well studied theory and numerical experiments [30]. On a similar note, the recent DiffGCN [7] also makes use of discretized operators to approximate the graph gradient and Laplacian. However, DiffGCN is specifically tailored for geometric tasks since it projects its components on the x, y, z axes, and is applied using a ResNet [15] (diffusion) structure only. Here we propose a network for both geometric and non-geometric tasks, and also utilize the diffusion or hyperbolic layer dynamics.

PDEs and CNNs: In a recent series of works, the connection between PDEs and CNNs was studied [19, 20, 21, 31, 32, 33, 34, 35]. It was shown that it is possible to treat a deep neural network as a dynamical system driven by some PDE, where each convolution layer is considered a time step of a discretized PDE. The connection between PDEs and CNNs was also used to reduce the computational burden [36]. Besides the interpretation of CNNs as a PDE solver, it was shown that it is also possible to learn a symbolic representation of PDEs in a task-and data driven approach in [37, 38].

3 Methods

3.1 Partial Differential Equations on manifolds

We show now that GCNs can be viewed as discretizations of PDEs on manifolds, similarly to CNNs that are viewed as discretized PDEs on regular grids [21]. Consider a general manifold $\mathcal{M}$ where a vector function f resides (also dubbed the features function), along with its continuous differential operators such as the gradient ∇, divergence $\nabla\cdot$ and the Laplacian Δ that reside on the manifold $\mathcal{M}$.

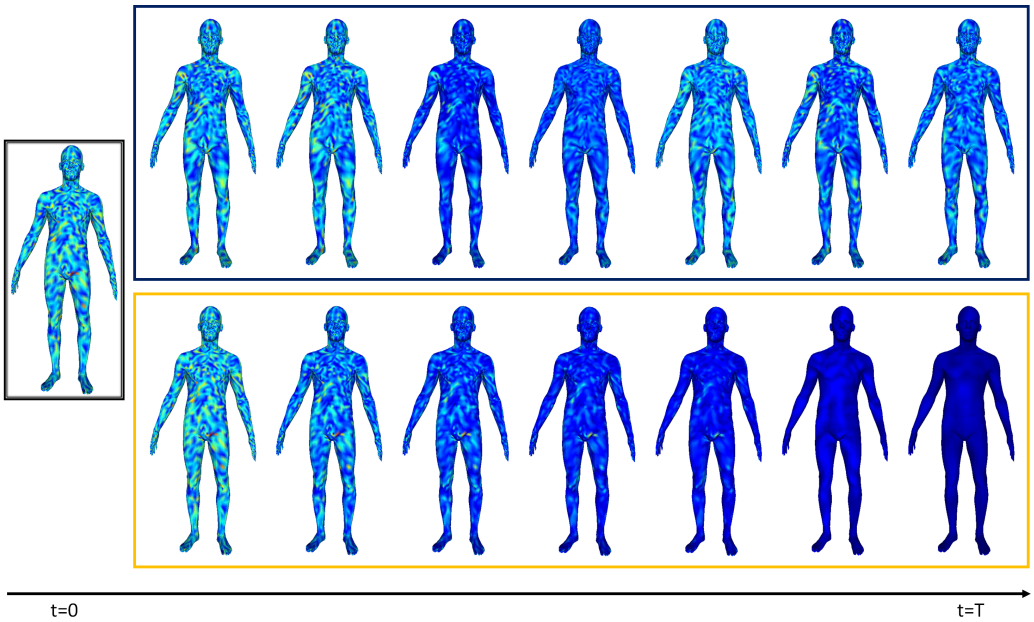


Figure 1: Feature evolvement on an input mesh (left). Propagation in time is from left to right. Hyperbolic and diffusion equation dynamics are on the top and bottom row, respectively. While a diffusive graph network (similar to most common GCNs that rely on ResNets) smooths the information on the manifold, the hyperbolic network yields a non-uniform field.

Given these differential operators, one can model different processes on $\mathcal{M}$. In particular, we consider two PDEs – the non-linear diffusion and the non-linear hyperbolic equations

$$f_t = \nabla \cdot K^* \sigma(K \nabla f), \quad f(t=0) = f^0, \quad t \in [0, T], \tag{1}$$

$$f_{tt} = \nabla \cdot K^* \sigma(K \nabla f), \quad f(t=0) = f^0, \quad f_t(t=0) = 0, \quad t \in [0, T], \tag{2}$$

respectively, equipped with appropriate boundary conditions. Here K is a coefficient matrix that can change in time and represents the propagation over the manifold $\mathcal{M}$, where K^* is its conjugate transpose and $\sigma(\cdot)$ is a non-linear activation function. Eq. (1) – (2) define a non-linear operator that takes initial feature vectors f^0 at time 0 and propagates them to time T, yielding f^T where they can be used for different tasks. We now provide two theorems that characterize the behavior of Eq. (1) – (2), based on ideas from [21][1].

Theorem 1. *If the activation function $\sigma(\cdot)$ is monotonically non-decreasing and sign-preserving, then the forward propagation through a diffusive PDE-GCN layer for $t \in [0, \infty)$ yields a non-increasing feature norm, that is,*

$$\frac{\partial}{\partial t} \|f\|^2 \leq 0.$$

Theorem 2. *Assume that the activation function $\sigma(\cdot)$ is monotonically non-decreasing, sign-preserving and satisfies $|\sigma(x)| \leq |x|$, and define the energy*

$$\mathcal{E}_{net} = \|f_t\|^2 + \left(K \nabla f, \sigma(K \nabla f)\right),$$

then $\mathcal{E}_{net} \leq c_K$, where c_K is a constant that depends on K but independent of time.

The outcome of those theorems is that the dynamics described in Eq. (1) is smoothing, while the one in Eq. (2) is bounded by a conserving mapping. An illustration of this behavior is presented in Fig. 1.

In the physical world, diffusion and hyperbolic equations are used for different applications. Similarly, many computational models for image segmentation [39], denoising [40] and deblurring are based on anisotropic diffusion which are similar to the model in Eq. (1). On the other hand, applications that require conservation such as volume/distance preservation as in the dense shape correspondence task [41] and protein folding [10], are typically better treated using a hyperbolic equation as in Eq. (2).

[1] See proofs in Appendix A.

3.2 Discretized differential operators on graphs

The models (1) and (2) reside in a continuous manifold $\mathcal{M}$, on which a continuous function vector f is defined. A graph can be thought of as a discretization of that manifold to a finite space. Assume we are given an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} \in \mathcal{M}$ is the set of n vertices of the graph and $\mathcal{E}$ is the set of m edges of the graph. Let us denote by $\mathbf{f}_i \in \mathbb{R}^c$ the value of the discrete version of f, on the i-th node of $\mathcal{G}$. c is the number of channels, which is the width of the neural network. We define $\mathbf{G}$, the discrete gradient operator on the graph, also known as the incidence matrix, as follows:

$$(\mathbf{G}\mathbf{f})_{ij} = \mathbf{W}_{ij}(\mathbf{f}_i - \mathbf{f}_j), \tag{3}$$

where nodes i and j are connected via the (i, j)-th edge, $\mathbf{W}_{ij}$ is an edge weight matrix which can be learnt, and $\mathbf{f}_i$ and $\mathbf{f}_j$ are the features on the i-th and j-th nodes, respectively. The gradient operator can be thought of as a weighted directional derivative of the function f in the direction defined by the nodes i and j. Furthermore, if we choose the scaled identity matrix $\mathbf{W}_{ij} = d_{ij}^{-1}\mathbf{I}$, where d_{ij} is the distance between the two nodes, then the discrete gradient is a second order approximation to the true gradient of the function f on the edges of the graph. In this work, we use $\mathbf{W}_{ij} = \gamma_{ij}\mathbf{I}$, where the scale γ_{ij} is the geometric mean of the degree of the nodes i, j. Note, that the gradient operator is a mapping from the *vertex* space to the *edge* space.

Given the gradient matrix, it is possible to define the divergence matrix [42], which is an approach that is used extensively in mimetic discretizations of PDEs. To this end, we define the inner product between an edge feature vector $\mathbf{q}$ and the gradient of a node feature vector $\mathbf{f}$ as

$$(\mathbf{q}, \mathbf{G}\mathbf{f}) = \mathbf{q}^\top \mathbf{G}\mathbf{f} = \mathbf{f}^\top \mathbf{G}^\top \mathbf{q}. \tag{4}$$

The divergence is naturally defined as the operator that maps edge operator $\mathbf{q}$ to the node space, that is $\nabla\cdot \approx -\mathbf{G}^\top$. As usual, the graph Laplacian operator can be obtained by taking the divergence of the gradient. In graph theory it is defined as a positive matrix that is, $\Delta \approx \mathbf{G}^\top\mathbf{G}$ [2].

We also define the weighted line integral over an edge. Similarly to Eq. (3) – (4), we define

$$(\mathbf{A}\mathbf{f})_{ij} = \frac{1}{2}\mathbf{W}_{ij}(\mathbf{f}_i + \mathbf{f}_j), \quad (\mathbf{q}, \mathbf{A}\mathbf{f}) = \mathbf{q}^\top \mathbf{A}\mathbf{f} = \mathbf{f}^\top \mathbf{A}^\top \mathbf{q}. \tag{5}$$

The operator $\mathbf{A}$ approximates the mass operator on the edges. The right equation in Eq. (5) suggests that an appropriate averaging operator for edge features is the transpose of the nodal edge average.

The advantage of defining such operators is that we are able to design networks with architectures that mimic the continuous operators (1) and (2) on the discrete level, as we show in the next section.

3.3 Graph Convolutional Networks by Partial Differential Equations

In order to use the computational models in Eq. (1) – (2), we form their discrete versions:

$$\mathbf{f}^{(l+1)} = \mathbf{f}^{(l)} - h\mathbf{G}^\top \mathbf{K}_l^\top \sigma(\mathbf{K}_l \mathbf{G}\mathbf{f}^{(l)}), \tag{6}$$

$$\mathbf{f}^{(l+1)} = 2\mathbf{f}^{(l)} - \mathbf{f}^{(l-1)} - h^2\mathbf{G}^\top \mathbf{K}_l^\top \sigma(\mathbf{K}_l \mathbf{G}\mathbf{f}^{(l)}). \tag{7}$$

Here, in Eq. (6) we use the forward Euler to discretize Eq. (1), and in Eq. (7) we discretize the second order time derivative in Eq. (2), using the leapfrog method. In both cases, $\mathbf{f}^{(l)}$ are the node features and $\mathbf{K}_l$ is a 1×1 trainable convolution of the l-th layer. The similarity between (6) and ResNet is well documented in the context of CNNs [21]. The hyper-parameter h is the step-size, and it is chosen such that the stability of the discretization is kept. We use $\sigma = \tanh$ for the activation function as it yields slightly better results in our experiments, although other functions such as ReLU can also be used, as reported in Sec. 4.7. Each of Eq. (6) – (7) defines a PDE-GCN block. We denote the former (diffusive equation) by PDE-GCN$_{\mathrm{D}}$ and the latter (hyperbolic equation) by PDE-GCN$_{\mathrm{H}}$.

To complete the description of our network, a few more details are required, as follows:

The (opening) embedding layer. The input vertex features $\mathbf{u}_\mathcal{V}$ are fed through an embedding (1×1 convolution) layer $\mathbf{K}_o$ to obtain the initial features $\mathbf{f}_0$ of our PDE-GCN network: $\mathbf{f}_0 = \mathbf{K}_o\mathbf{u}_\mathcal{V}$. In cases where input edge attributes (features) $\mathbf{u}_\mathcal{E}$ are available (as in the experiment in Sec. 4.6), we

[2] In the field of numerical PDEs, the Laplacian is defined as $-\mathbf{G}^\top\mathbf{G}$, but the combinatorial Laplacian is defined as $\mathbf{G}^\top\mathbf{G}$.

transform them to the vertex space by taking their divergence and average, and concatenate them to the input of the embedding layer $\mathbf{K}_o$ as follows: $\mathbf{f}_0 = \mathbf{K}_o(\mathbf{u}_\mathcal{V} \oplus \mathbf{A}^\top \mathbf{u}_\mathcal{E} \oplus \mathbf{G}^\top \mathbf{u}_\mathcal{E})$.

The (closing) embedding layer. Given the final vertex features $\mathbf{f}^{(L)}$ of our PDE-GCN with L layers, we obtain the output of the network by performing: $\mathbf{u}_{out} = \mathbf{K}_c \mathbf{f}^{(L)}$. Here $\mathbf{K}_c$ is a 1×1 convolution layer mapping the hidden feature space to the output shape.

Initialization. The 1×1 convolutions $\mathbf{K}_l$ in Eq. (6) – (7) are initialized as identity. This way, the network begins as a diffusion/hyperbolic equation and is further improved by the learning process.

The choice of dynamics. For some applications, anisotropic diffusion is appropriate, while for others conservation is more important. However, in some applications this may not be clear. To this end, it is possible to combine Eq. (1) – (2) to obtain the continuous process

$$\alpha f_{tt} + (1-\alpha) f_t = \nabla \cdot K^* \sigma(K \nabla f), \quad f(t=0) = f^0, \quad f_t(t=0) = 0 \quad t \in [0, T], \tag{8}$$

where $\alpha = sigmoid(\beta)$, meaning $0 \leq \alpha \leq 1$, and β is a single trainable parameter. The discretization of this PDE leads to the following network dynamics:

$$\alpha(\mathbf{f}^{(l+1)} - 2\mathbf{f}^{(l)} + \mathbf{f}^{(l-1)}) + h(1-\alpha)(\mathbf{f}^{(l+1)} - \mathbf{f}^{(l)}) = -h^2 \mathbf{G}^\top \mathbf{K}_l^\top \sigma(\mathbf{K}_l \mathbf{G} \mathbf{f}^{(l)}), \tag{9}$$

where $\mathbf{f}^{(l+1)}$ is updated by the known $\mathbf{f}^{(l)}$ and $\mathbf{f}^{(l-1)}$. Note, that it is also possible to learn a combination coefficient α_i per layer, although we did not read a benefit from such scheme.

As we show in our numerical experiments, learning α yields results that are consistent with our understanding of the problem, that is, graph node-classification gravitates towards no second order derivatives while applications that require conservation gravitate towards the hyperbolic equation.

4 Experiments

In this section we demonstrate our approach on various problems from different fields and applications ranging from 3D shape-classification [43] to protein-protein interaction [25] and node-classification [44]. The experiments also vary in their output type. Node classification is similar to segmentation problems that are typically solved by anisotropic diffusion while the dense shape correspondence problem is conservative and therefore can be thought of as a problem that does not require smoothing.

In all experiments, we use the suitable PDE-GCN (D or H) block as described in Sec. 3, with various depths (number of layers) and widths (number of channels), as well as the appropriate final convolution steps, depending on the task at hand. A detailed description of the architectures used in our experiments is given in Appendix B. We use the Adam [45] optimizer in all experiments, and perform grid search over the hyper-parameters of our network. The selected hyper-parameters are reported in Appendix C. Our 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 [46], trained on an Nvidia Titan RTX GPU.

We show that for all the considered tasks and datasets, our method is either remarkably better or on par with state-of-the-art models.

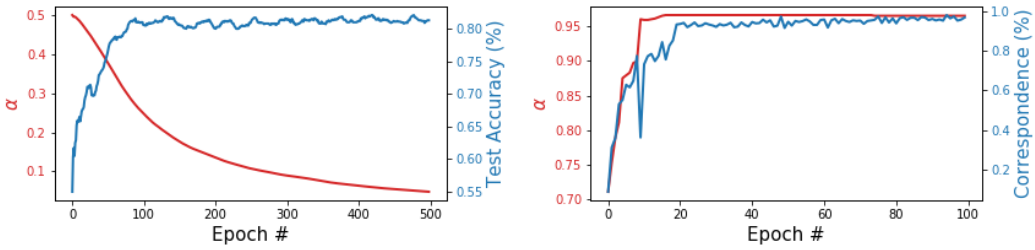
4.1 Generalization of GCNs to different applications

To gain deeper understanding about the effectiveness and generalization of various GCN methods to different tasks, we start by picking two datasets - ModelNet-10 [43] for 3D shape-classification, and Cora [44] for semi-supervised node classification. Those datasets are not only different in terms of application (global versus local classification), but also stem from different domains. While in ModelNet-10 the data has geometrical meaning, the data in Cora has no obvious geometrical interpretation. Therefore, we suggest that success in both applications should be obtained from a generalizable GCN. We compare our PDE-GCN$_\mathrm{D}$ with two recent and popular networks - DGCNN [5] and GCNII [18]. For ModelNet-10 shape-classification, we randomly sample 1,024 points from each shape to form a point cloud, and connect its points using k-nearest-neighbors (k-nn) algorithm with $k = 10$ to obtain a graph and follow the training scheme of [5]. On Cora, we follow the same procedure as in [18]. We evaluate all models with 4 layers, as well as 2 layers for DGCNN on Cora.

Our results, reported in Tab. 1 suggest that while each of the considered methods obtains high accuracy on the dataset it originally was tested on (Cora for GCNII and ModelNet-10 for DGCNN),

Table 1: Generalization of GCNs to different domains and applications. (L) denotes L layers.

Method	Dataset	Accuracy (%)
DGCNN (4) [5]	ModelNet-10	92.8
DGCNN (2) / (4)	Cora	34.9 / 25.2
DGCNN + Diffusion (2) / (4)	Cora	71.0 / 66.1
GCNII (4) [18]	ModelNet-10	65.4
	Cora	82.6
PDE-GCN $_{\mathrm{D}}$(4) (Ours)	ModelNet-10	92.2
	Cora	83.6



(a) Graph node-classification (Cora)

(b) Dense shape correspondence (FAUST)

Figure 2: Learnt mixture of the hyperbolic equation α, to the dynamics of the network. The diffusion equation contribution is $1 - \alpha$.

obtaining a similar measure of success on a different dataset was not possible when using the very same networks. Additionally, while our attempts to add a diffusion-equation dynamics to DGCNN (i.e., updating features as in Eq. (6)) showed an increase in performance – a large gap to state-of-the-art model still exists. On top of that, we also see that DGCNN suffers from over-smoothing, as its accuracy significantly decreases when adding more layers. Last but not least, we observe that our PDE-GCN$_{\mathrm{D}}$ obtains high accuracy on both datasets, similar or better than state-of-the-art models.

4.2 Learning PDE network dynamics

In this experiment, we delve on the ability to *learn* the appropriate PDE that better models a given problem. To this end, we use the mixture model from Eq. (9) so that the resulting PDE is a combination of the diffusion and hyperbolic dynamics. We use a 8 layer mixed PDE-GCN, starting with $\alpha = 0.5$, such that it is balanced between a PDE-GCN$_{\mathrm{D}}$ and PDE-GCN$_{\mathrm{H}}$. By learning the parameter α in (9), we allow to choose a mixed PDE between a purely conservative network and a diffusive one. We consider two problems: semi-supervised node classification on Cora, and dense shape correspondence on FAUST [47].

Our results, reported in Fig. 2 suggest that just as in classical works [39, 48], problems like node-classification obtain better performance with an anisotropic diffusion like in Eq. (6), and for problems involving dense-correspondences like in [41, 10] that tend to conserve the energy of the underlying problem, a hyperbolic equation type of PDE as in Eq. (7) is more appropriate.

Table 2: Statistics of datasets used in our semi-and fully supervised node-classification experiments.

Dataset	Cora	CiteSeer	PubMed	Chameleon	Cornell	Texas	Wisconsin	PPI
Classes	7	6	3	5	5	5	5	121
Nodes	2,708	3,327	19,717	2,277	183	183	251	56,944
Edges	5,429	4,732	44,338	36,101	295	309	499	818,716
Features	1,433	3,703	500	2,325	1,703	1,703	1,703	50

4.3 Semi-supervised node classification

In this set of experiments we use three datasets – Cora, Citeseer and PubMed [44]. For all datasets we use the standard training/validation/testing split as in [49], with 20 nodes per class for training, 500 validation nodes and 1,000 testing nodes and follow the training scheme of [18]. The statistics of the datasets are reported in Tab. 2. We compare our results using PDE-GCN$_{\mathrm{D}}$ with recent and popular models like GCN [3], GAT [26], APPNP [29], JKNet [28] and DropEdge [27]. We note that our network does not over-smooth, as an increase in the number of layers does not cause performance degradation. For example, on CiteSeer, we obtain 75.6% accuracy with 32 layers, compared to 74.6% with two layers[3]. Overall, our results in Tab. 3 show that our PDE-GCN$_{\mathrm{D}}$ achieves similar or better accuracy than the considered methods.

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

		Layers					
Dataset	Method	2	4	8	16	32	64
Cora	GCN [3]	**81.1**	80.4	69.5	64.9	60.3	28.7
	GCN (Drop) [27]	**82.8**	82.0	75.8	75.7	62.5	49.5
	JKNet [28]	–	80.2	80.7	80.2	**81.1**	71.5
	JKNet (Drop) [27]	–	**83.3**	82.6	83.0	82.5	83.2
	Incep [27]	–	77.6	76.5	81.7	**81.7**	80.0
	Incep (Drop) [27]	–	82.9	82.5	83.1	83.1	**83.5**
	GCNII [18]	82.2	82.6	84.2	84.6	85.4	**85.5**
	GCNII* [18]	80.2	82.3	82.8	83.5	84.9	**85.3**
	PDE-GCN$_{\mathrm{D}}$ (Ours)	82.0	83.6	84.0	84.2	84.3	**84.3**
Citeseer	GCN [3]	**70.8**	67.6	30.2	18.3	25.0	20.0
	GCN (Drop) [27]	**72.3**	70.6	61.4	57.2	41.6	34.4
	JKNet [28]	–	68.7	67.7	**69.8**	68.2	63.4
	JKNet (Drop) [27]	–	72.6	71.8	**72.6**	70.8	72.2
	Incep [27]	–	69.3	68.4	**70.2**	68.0	67.5
	Incep (Drop) [27]	–	**72.7**	71.4	72.5	72.6	71.0
	GCNII [18]	68.2	68.8	70.6	72.9	**73.4**	73.4
	GCNII* [18]	66.1	66.7	70.6	72.0	**73.2**	73.1
	PDE-GCN$_{\mathrm{D}}$ (Ours)	74.6	75.0	75.2	75.5	**75.6**	75.5
Pubmed	GCN [3]	**79.0**	76.5	61.2	40.9	22.4	35.3
	GCN (Drop) [27]	**79.6**	79.4	78.1	78.5	77.0	61.5
	JKNet [28]	–	78.0	**78.1**	72.6	72.4	74.5
	JKNet (Drop) [27]	–	78.7	78.7	**79.7**	79.2	78.9
	Incep [27]	–	77.7	**77.9**	74.9	–	–
	Incep (Drop) [27]	–	**79.5**	78.6	79.0	–	–
	GCNII [18]	78.2	78.8	79.3	**80.2**	79.8	79.7
	GCNII* [18]	77.7	78.2	78.8	**80.3**	79.8	80.1
	PDE-GCN$_{\mathrm{D}}$ (Ours)	79.3	**80.6**	80.1	80.4	80.2	80.3

4.4 Fully-supervised node classification

We follow [50] and use 7 datasets: Cora, CiteSeer, PubMed, Chameleon, Cornell, Texas and Wisconsin. We also use the same train/validation/test splits of $60\%, 20\%, 20\%$, respectively. In addition, we report the average performance over 10 random splits from [50]. We fix the number of channels to 64 and perform grid search to determine the hyper-parameters parameters, which are reported in Appendix C. We compare our network with GCN, GAT, three variants of Geom-GCN [50], APPNP, JKNet, Incep and GCNII in Tab. 4. Our experiments read either similar or better than the state-of-the-art on Cora, CiteSeer and PubMed datasets. On Chameleon, Cornell, Texas and Wisconsin datasets, we improve state-of-the-art accuracy by a significant margin. For example, we

[3]Note that this result is also a new state-of-the-art accuracy.

Table 4: Fully-supervised node classification accuracy (%). (L) indicates a L layers network.

Method	Cora	Cite.	Pubm.	Cham.	Corn.	Texas	Wisc.
GCN [3]	85.77	73.68	88.13	28.18	52.70	52.16	45.88
GAT [26]	86.37	74.32	87.62	42.93	54.32	58.38	49.41
Geom-GCN-I [50]	85.19	77.99	90.05	60.31	56.76	57.58	58.24
Geom-GCN-P [50]	84.93	75.14	88.09	60.90	60.81	67.57	64.12
Geom-GCN-S [50]	85.27	74.71	84.75	59.96	55.68	59.73	56.67
APPNP [29]	87.87	76.53	89.40	54.30	73.51	65.41	69.02
JKNet [28]	85.25 (16)	75.85 (8)	88.94 (64)	60.07 (32)	57.30 (4)	56.49 (32)	48.82 (8)
JKNet (Drop) [27]	87.46 (16)	75.96 (8)	89.45 (64)	62.08 (32)	61.08 (4)	57.30 (32)	50.59 (8)
Incep (Drop) [27]	86.86 (8)	76.83 (8)	89.18 (4)	61.71 (8)	61.62 (16)	57.84 (8)	50.20 (8)
GCNII [18]	88.49 (64)	77.08 (64)	89.57 (64)	60.61 (8)	74.86 (16)	69.46 (32)	74.12 (16)
GCNII*	88.01 (64)	77.13 (64)	**90.30** (64)	62.48 (8)	76.49 (16)	77.84 (32)	81.57 (16)
PDE-GCN$_{\mathrm{D}}$ (Ours)	**88.51** (16)	**78.36** (64)	89.6 (64)	**64.12** (8)	89.19 (2)	90.81 (8)	90.39 (8)
PDE-GCN$_{\mathrm{H}}$ (Ours)	87.71 (32)	78.13 (16)	89.16 (16)	61.57 (64)	**89.45** (64)	**92.16** (64)	**91.37** (16)

Table 5: Protein-protein interaction (PPI). Results are reported in micro-averaged F1 score.

Method	Micro-averaged F1
GraphSAGE [25]	61.20
VR-GCN [51]	97.80
GaAN [52]	98.71
GAT [26]	97.30
JKNet [28]	97.60
GeniePath [53]	98.50
Cluster-GCN [54]	99.36
GCNII [18]	99.54
GCNII* [18]	**99.58**
PDE-GCN$_{\mathrm{D}}$ (Ours)	99.07

Table 6: Dense shape correspondence (%) with zero geodesic error

Method	Faust
ACNN [55]	63.8
MoNet [4]	89.1
FMNet [56]	98.2
SplineCNN [57]	99.2
PDE-GCN$_{\mathrm{D}}$ (Ours)	64.2
PDE-GCN$_{\mathrm{H}}$ (Ours)	**99.9**

obtain 92.16% accuracy on Texas with our PDE-GCN$_{\mathrm{H}}$, compared to 77.84% with GCNII*. Similar improvements hold for Cornell and Wisconsin datasets. The common factor for these datasets is their small size, as depicted from Tab. 2. We argue that the success of our network stems from its capability of apriori extracting features from graphs, due to its utilization of discretized differential operators and PDE guided construction. On Chameleon, using PDE-GCN$_{\mathrm{D}}$, we improve the current state-of-the-art accuracy of GCNII* from 62.48% to 64.12%. Also, we note that unlike in the semi-supervised case, where some of the labels are missing, here it is possible to obtain meaningful results with the hyperbolic equation based PDE-GCN$_{\mathrm{H}}$ as we do not have unknown nodes in the fully-supervised case, which would be otherwise preserved using the hyperbolic-equation dynamics.

4.5 Inductive Learning

We follow [18] and employ the PPI dataset [25] for the inductive learning task. We use a 8 layer PDE-GCN$_{\mathrm{H}}$ network, without dropout or weight-decay, and a learning rate of 0.001. We compare our results with methods like GraphSAGE, GAT, JKNet, GeniePath, GCNII and others. As reported in Tab. 5, our PDE-GCN $_{\mathrm{D}}$ achieves 99.07 Micro-averaged F1 score, superior to methods like GAT, JKNet and GeniePath, also close to state-of-the-art GCNII* with a score of 99.58.

4.6 Dense shape correspondence

Finding dense correspondences between shapes is a classical experiment for hyperbolic dynamics, as we are interested in modeling local motion dynamics. In essence, learning to find correspondences between shapes is similar to learning a transformation from one shape to the other. To this end, we use the FAUST dataset [47] containing 10 scanned human shapes in 10 different poses with 6,890

Table 7: Ablation study of PDE-GCN accuracy ($\%$) on semi-supervised node-classification.

Method	Dataset	Layers					
		2	4	8	16	32	64
PDE-GCN$_{\mathrm{H}}$	Cora	79	79.3	78.0	78.0	77.8	77.5
	CiteSeer	70.9	71.7	72.1	72.3	72.5	72.4
PDE-GCN$_{\mathrm{D}}$ (non-symmetric)	Cora	83.5	83.3	83.6	83.1	82.7	81
	CiteSeer	74.3	74.5	74.8	75.0	73.9	73.3
PDE-GCN$_{\mathrm{D}}$ ($\sigma = ReLU$)	Cora	80.3	81.8	82.6	83.0	83.4	83.5
	CiteSeer	73.1	73.2	72.8	73.3	73.6	74.0

nodes each. We follow the train and test split from [4], where the first 80 subjects are used for training and the remaining 20 subjects for testing. Our metric is the correspondence percentage with zero geodesic error, i.e., the percentage of perfectly matched shapes from all our test cases. We follow the pre-processing and training scheme of [57] where we use Cartesian coordinates to describe distances between nodes, with initial features of a constant $\mathbf{1} \in \mathbb{R}^n$ where n is the number of nodes. We use a 8 layer PDE-GCN$_{\mathrm{D}}$ and PDE-GCN$_{\mathrm{H}}$ variants, both with constant learning rate of 0.001 with no weight-decay or dropout, and compare to recent and popular methods like ACNN [55], MoNet [4], FMNet [56], and SplineCNN [57]. As expected from the discussion in Sec. 3.1, and reported in Tab. 6, the hyperbolic equation proves to be a better fit for this kind of problem. Also, our PDE-GCN$_{\mathrm{H}}$ achieves a promising 99.9% correspondence rate with zero geodesic error, outperforming the rest of the considered methods.

4.7 Ablation study

Our method has two main dynamics – diffusion and hyperbolic. To verify our proposal in Sec. 3.1, we examine the performance of the hyperbolic PDE-GCN$_{\mathrm{H}}$ on semi-supervised node-classification on Cora and CiteSeer datasets. Our results in Tab. 7 show that indeed for problems where we wish to obtain a piecewise-constant prediction, the diffusive formulation of our network, PDE-GCN$_{\mathrm{D}}$, is more suitable. Furthermore, we study the importance of the positive-semi definiteness of our learnt operator as described in Eq. (6) – (7), by removing the $\mathbf{K}_l^T$ term from the dynamics equations. This yields a non-symmetric operator that does not guarantee positive-semi-definiteness. We note that the enforcement of the latter is important to obtain higher accuracy, which is improved by up to 3% with the introduction of a positive semi-definite operator. Also, we report on the use of $\sigma = ReLU$, from the discussion in Sec. 3.3, where we favor $\tanh$ as an activation function.

5 Summary

In this paper we explored new architectures for graph neural networks. Our motivation stems from the similarities between graph networks and time dependent partial differential equations that are discretized on manifolds and graphs. By adopting an appropriate PDE, and embedding the finite graph in an infinite manifold, we are able to define networks that are either diffusive, conservative, or a combination of both.

Not all natural phenomena are solved using the same PDE and we should not expect that all graph problems should be solved by the same network dynamics. To this end we allow the data to choose which type of network is appropriate for the solution of the problem (diffusive or hyperbolic). Indeed, numerical experiments show that the network gravitates towards a hyperbolic one for problems where conservation is required, and towards a diffusive one when anisotropic diffusion is favorable.

Finally, we showed that the proposed networks can be made deep without over-smoothing and, can deliver the state-of-the-art performance or improve it for virtually every problem we worked with. In particular, our network dramatically improved the state-of-the-art for problems that are data-poor. We believe that for such problems the structure imposed by our dynamics and operators regularizes the network and therefore yields implicit regularization.

References

[1] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[2] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852, 2016.

[3] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[4] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5115–5124, 2017.

[5] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *arXiv preprint arXiv:1801.07829*, 2018.

[6] Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. Meshcnn: a network with an edge. *ACM Transactions on Graphics (TOG)*, 38(4):90, 2019.

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

[8] Alexey Strokach, David Becerra, Carles Corbi-Verge, Albert Perez-Riba, and Philip M. Kim. Fast and flexible protein design using deep graph neural networks. *Cell Systems*, 11(4):402 – 411.e4, 2020.

[9] Thomas Gaudelet, Ben Day, Arian R Jamasb, Jyothish Soman, Cristian Regep, Gertrude Liu, Jeremy BR Hayter, Richard Vickers, Charles Roberts, Jian Tang, et al. Utilising graph machine learning within drug discovery and development. *arXiv preprint arXiv:2012.05716*, 2020.

[10] Moshe Eliasof, Tue Boesen, Eldad Haber, Chen Keasar, and Eran Treister. Mimetic neural networks: A unified framework for protein design and folding. *arXiv preprint arXiv:2102.03881*, 2021.

[11] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 974–983, 2018.

[12] Jiezhong Qiu, Jian Tang, Hao Ma, Yuxiao Dong, Kuansan Wang, and Jie Tang. Deepinf: Social influence prediction with deep learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2110–2119, 2018.

[13] Chang Li and Dan Goldwasser. Encoding social information with graph convolutional networks forpolitical perspective detection in news media. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2594–2604, 2019.

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

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[16] Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnns. In *International Conference on Learning Representations*, 2020.

[17] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34:3438–3445, 04 2020.

[18] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1725–1735. PMLR, 13–18 Jul 2020.

[19] Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *Inverse Problems*, 34(1), 2017.

[20] Bo Chang, Lili Meng, Eldad Haber, Lars Ruthotto, David Begert, and Elliot Holtham. Reversible architectures for arbitrarily deep residual neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[21] Lars Ruthotto and Eldad Haber. Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision*, pages 1–13, 2019.

[22] Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3693–3702, 2017.

[23] Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 37–45, 2015.

[24] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org, 2017.

[25] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, 2017.

[26] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018.

[27] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2020.

[28] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5453–5462. PMLR, 10–15 Jul 2018.

[29] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Combining neural networks with personalized pagerank for classification on graphs. In *International Conference on Learning Representations*, 2019.

[30] Uri M Ascher. *Numerical methods for evolutionary differential equations*. SIAM, 2008.

[31] E Weinan. A Proposal on Machine Learning via Dynamical Systems. *Communications in Mathematics and Statistics*, 5(1):1–11, March 2017.

[32] Pratik Chaudhari, Adam Oberman, Stanley Osher, Stefano Soatto, and Guillaume Carlier. Deep relaxation: partial differential equations for optimizing deep neural networks. *Research in the Mathematical Sciences*, 5(3):30, 2018.

[33] Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In *International Conference on Machine Learning (ICML)*, 2018.

[34] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pages 6571–6583, 2018.

[35] Eldad Haber, Keegan Lensink, Eran Triester, and Lars Ruthotto. Imexnet: A forward stable deep neural network. *arXiv preprint arXiv:1903.02639*, 2019.

[36] Jonathan Ephrath, Moshe Eliasof, Lars Ruthotto, Eldad Haber, and Eran Treister. Leanconvnets: Low-cost yet effective convolutional neural networks. *IEEE Journal of Selected Topics in Signal Processing*, 2020.

[37] Yohai Bar-Sinai, Stephan Hoyer, Jason Hickey, and Michael P. Brenner. Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences*, 116(31):15344–15349, 2019.

[38] Zichao Long, Yiping Lu, and Bin Dong. Pde-net 2.0: Learning pdes from data with a numeric-symbolic hybrid deep network. *Journal of Computational Physics*, page 108925, 2019.

[39] Vicent Caselles, Ron Kimmel, and Guillermo Sapiro. Geodesic active contours. *International journal of computer vision*, 22(1):61–79, 1997.

[40] Leonid I Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1):259–268, 1992.

[41] Mathieu Aubry, Ulrich Schlickewei, and Daniel Cremers. The wave kernel signature: A quantum mechanical approach to shape analysis. In *2011 IEEE international conference on computer vision workshops (ICCV workshops)*, pages 1626–1633. IEEE, 2011.

[42] J Hyman, J Morel, M Shashkov, and Stanly Steinberg. Mimetic finite difference methods for diffusion equations. *Computational Geosciences*, 6(3):333–352, 2002.

[43] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.

[44] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[45] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[46] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[47] Federica Bogo, Javier Romero, Matthew Loper, and Michael J. Black. FAUST: Dataset and evaluation for 3D mesh registration. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Piscataway, NJ, USA, June 2014. IEEE.

[48] Ross T Whitaker. A level-set approach to 3d reconstruction from range data. *International journal of computer vision*, 29(3):203–231, 1998.

[49] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pages 40–48. PMLR, 2016.

[50] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. In *International Conference on Learning Representations*, 2020.

[51] Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 942–950. PMLR, 10–15 Jul 2018.

[52] Jiani Zhang, Xingjian Shi, Junyuan Xie, Hao Ma, Irwin King, and Dit-Yan Yeung. Gaan: Gated attention networks for learning on large and spatiotemporal graphs. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence*, pages 339–349, 2018.

[53] Ziqi Liu, Chaochao Chen, Longfei Li, Jun Zhou, Xiaolong Li, and Le Song. Geniepath: Graph neural networks with adaptive receptive paths. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33, 02 2018.

[54] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 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.

[55] Davide Boscaini, Jonathan Masci, Emanuele Rodolà, and Michael Bronstein. Learning shape correspondence with anisotropic convolutional neural networks. 05 2016.

[56] Or Litany, Tal Remez, Emanuele Rodolà, Alex Bronstein, and Michael Bronstein. Deep functional maps: Structured prediction for dense shape correspondence. pages 5660–5668, 10 2017.

[57] Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Heinrich Müller. Splinecnn: Fast geometric deep learning with continuous b-spline kernels, 2017.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] Our theorems state that the activation function must be sign-preserving, bounded and monotone.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] We did not find any societal impacts of the work.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] Proofs are provided in the supplementary material.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] The code and the links to the (public) data sets are provided in the supplementary material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]