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

**Anonymous Author(s)**
Affiliation
Address
`email`

## Abstract

Graph neural networks are have shown their efficacy in fields such as computer vision, computational biology and chemistry, where data are naturally explained by graphs. However, unlike convolutional neural networks, deep graph networks do not necessarily yield better performance than shallow networks. This behaviour usually stems from the over-smoothing phenomenon. In this work, we propose a family of architectures to control this behaviour by design. Our networks are motivated by numerical methods for solving Partial Differential Equations (PDEs) on manifolds, and as such, their behaviour can be explained by similar analysis.

## 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, 8] to computational biology [9, 10, 11], recommendation systems [12] and social network analysis [13, 14]. However, GNNs still suffer from being typically *shallow*, as opposed to the concept of deep convolutional neural networks (CNNs) [15, 16]. This shortcoming is due to the *over-smoothing* phenomenon [17, 18, 19], where the node feature vectors become almost identical, such that they are indistinguishable, which yields non-optimal performance. Furthermore, because many GCNs lack theoretical guarantees, it is difficult to reason about their behaviour. These observations motivate us to develop a profound understanding of GNNs 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 [20, 21, 22]. 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 [22], we adopt this interpretation to explore versions of GCNs as PDEs on graphs or manifolds. We therefore call our network architectures *PDE-GCN*.

## 2 Related work

**Graph Convolutional Networks:** GCNs are typically divided into spectral [1, 3, 2] and spatial [23, 24, 25, 4] categories. Most of those can be implemented using the Message-Passing Neural Network paradigm [25], where each node aggregates features (messages) from its neighbours, according to some scheme. The works [3, 2] use polynomials of the graph Laplacian to parameterize the convolution operator. Works like and [26, 27, 28] propose methods to learn the propagation operators in GNNs instead of a Laplacian based operator as previously discussed.

Several of the methods above suffer from over-smoothing [17, 19], leading to undesired node features similarity for deep networks. To overcome this problem, some approaches rely on imposing regularization and augmentation as in PairNorm [17] and DropEdge [29], while other methods propose by dedicated construction [30, 19] In our work we construct a network that inherently does not over-smooth, based on discretized PDEs. Hence, we are able to motivate our choices by well studied theory and numerical experiments [31]. On a similar note, the recent DiffGCN [8] also makes use of discretized operators. However, DiffGCN is specific for geometric tasks. Also, GRAND [32] applies attention mechanism with diffusive dynamics, using several integration schemes. Here we propose a network that utilizes the diffusion or hyperbolic layer dynamics and their learnt mixture.

**PDEs and CNNs:** In a recent series of works, the connection between PDEs and CNNs was studied [20, 21, 22, 33, 34, 35, 36, 37]. 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 PDE. The connection between PDEs and CNNs was also used to reduce the computational burden [38], and here we harness PDE concepts to design and construct GCNs for a variety of applications.

## 3 Method

### 3.1 Partial Differential Equations on manifolds

Consider a general manifold $\mathcal{M}$ where a vector function $f$ resides, along with its continuous differential operators such as the gradient $\nabla$, divergence $\nabla\cdot$ and the Laplacian $\Delta$. 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 $\mathcal{M}$, $K^*$ 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 $f^T$ that can be used for different tasks. We now provide two theorems that characterize Eq. (1)-(2), based on ideas from [22][1].

**Theorem 1.** *If the activation function $\sigma(\cdot)$ is monotonically non-decreasing and sign-preserving, then the forward propagation through the diffusive PDE in (1) 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 energy*

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

*then the forward propagation through the hyperbolic PDE in (2) satisfies $\mathcal{E}_{net} \leq c_K$, 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. We demonstrate this behaviour in Fig. 1.

Many computational models for image segmentation [39], denoising [40] and deblurring are based on anisotropic diffusion, similar to the model in Eq. (1). On the other hand, applications that require volume/distance preservation as in the dense shape correspondence task [41] and protein folding [11], are typically better treated using a hyperbolic equation as in Eq. (2). Those insights motivate us to construct GCN layers according to Eq. (1)-(2) using discretized differential operators on graphs.

### 3.2 Discretized differential operators on graphs

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

[1]See proofs in Appendix A.

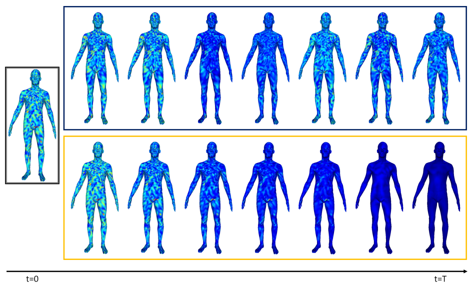

Figure 1: Feature evolvement on a mesh (left). Propagation in time is from left to right. Hyperbolic and diffusion equation dynamics are on the top and bottom row, respectively. While a diffusive layer smoothes the information on the manifold, the hyperbolic layer yields a non-uniform field.

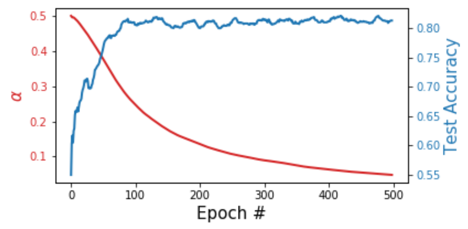

Figure 2: Graph node-classification.

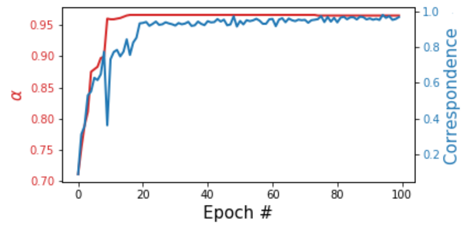

Figure 3: Dense shape correspondence.

$m$ 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. 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]. 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, $\Delta \approx \mathbf{G}^\top\mathbf{G}$

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.

### 3.3 PDE-GCN: 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 $1 \times 1$ trainable convolution of the $l$-th layer. 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.Each of Eq. (6)-(7) defines a PDE-GCN layer. We denote the former by PDE-GCN$_\mathrm{D}$ and the latter by PDE-GCN$_\mathrm{H}$.

**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

Table 1: 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 [45] | 85.27 | 77.99 | 90.05 | 60.90 | 60.81 | 67.57 | 64.12 |
| APPNP [46] | 87.87 | 76.53 | 89.40 | 54.30 | 73.51 | 65.41 | 69.02 |
| JKNet [30] | 85.25 (16) | 75.85 (8) | 88.94 (64) | 60.07 (32) | 57.30 (4) | 56.49 (32) | 48.82 (8) |
| JKNet (Drop) [29] | 87.46 (16) | 75.96 (8) | 89.45 (64) | 62.08 (32) | 61.08 (4) | 57.30 (32) | 50.59 (8) |
| Incep (Drop) [29] | 86.86 (8) | 76.83 (8) | 89.18 (4) | 61.71 (8) | 61.62 (16) | 57.84 (8) | 50.20 (8) |
| GCNII [19] | 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) |
| PDE-GCN$_{\mathrm{M}}$ (Ours) | **88.60** (16) | **78.48** (32) | 89.93 (16) | **66.01** (16) | **89.73** (64) | **93.24** (32) | **91.76** (16) |

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 $\beta$ is a single trainable parameter. The discretization of this PDE leads to the following network dynamics:

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

We denote a layer that is governed by Eq. (9) by PDE-GCN$_{\mathrm{M}}$.

## 4 Experiments

In this section we demonstrate our approach by first showing that it is possible to learn the desired PDE-dynamics. Then, we show the efficacy of our PDE-GCN on 7 real-world fully-supervised node classification datasets. A detailed description of the architectures and hyper-parameters used in our experiments are given in Appendicies C–D.

**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 $\alpha$ 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 [43].

Our results, reported in Fig. 2–3 suggest that just as in classical works [39, 44], problems like node-classification obtain better performance with an anisotropic diffusion like in Eq. (6), and for problems involving dense-correspondences like in [41, 11] that tend to conserve the energy of the underlying problem, a hyperbolic equation type of PDE as in Eq. (7) is more appropriate.

**Fully-supervised node classification** We follow [45] 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. We report the average performance over 10 random splits from [45]. Our results in Tab. 1 read either similar or better than the state-of-the-art models.

## 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. We showed that the proposed networks can be made deep without over-smoothing and can deliver the state-of-the-art performance.

## 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] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9267–9276, 2019.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

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

[18] 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.

[19] 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.

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

[21] 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.

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

[23] 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.

[24] 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.

[25] 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.

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

[27] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*, 2021.

[28] Moshe Eliasof, Eldad Haber, and Eran Treister. pathgcn: Learning general graph spatial operators from paths. In *International Conference on Machine Learning*, pages 5878–5891. PMLR, 2022.

[29] 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.

[30] 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.

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

[32] Benjamin Paul Chamberlain, James Rowbottom, Maria Gorinova, Stefan Webb, Emanuele Rossi, and Michael M Bronstein. Grand: Graph neural diffusion. *arXiv preprint arXiv:2106.10934*, 2021.

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

[34] 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.

[35] 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.

[36] 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.

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

[38] 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.

[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] 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.

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

[45] 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.

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

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

[48] 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.

## A Theorems and proofs

We repeat the theorems presented in Sec. 3 and provide their proofs below. The theorems hold for Neumann boundary conditions, which we use in our implementation—this is achieved by the construction of the differential operators. The proofs follow the ones presented in [22].

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

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

*Proof.* Let us examine the following inner product following Eq. (1):

$$(f, f_t) = (f, \nabla \cdot K^* \sigma(K\nabla f))$$

From integration by parts it holds that :

$$\frac{1}{2}\frac{\partial}{\partial t}\|f\|^2 = -(\nabla f, K^* \sigma(K\nabla f)) = -(K\nabla f, \sigma(K\nabla f)).$$

Plugging the definition of an inner product, together with the assumption that $\sigma$ is a sign-preserving function, it follows that:

$$sign(K\nabla f) = sign(\sigma(K\nabla f)).$$

Therefore, the following is non-positive:

$$\frac{1}{2}\frac{\partial}{\partial t}\|f\|^2 = -(K\nabla f, \sigma(K\nabla f)) \leq 0$$

Meaning

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

□

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

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

*then the forward propagation through the hyperbolic PDE in* (2) *satisfies $\mathcal{E}_{net} \leq c_K$, where $c_K$ is a constant that depends on $K$ but independent of time.*

*Proof.* Let us define the following energy:

$$\mathcal{E}_{lin} = \|f_t\|^2 + (K\nabla f, K\nabla f)$$

This energy is associated with the linear hyperbolic (wave-like) equation:

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

Assuming $K$ is constant in time, we obtain:

$$\frac{1}{2}\partial_t \mathcal{E}_{lin} = (f_t, f_{tt} - \nabla \cdot K^* K\nabla f) = 0$$

This means that the energy $\mathcal{E}_{lin}$ is constant in time, i.e. there exists some $c_K$ such that $\mathcal{E}_{lin} = c_K$.

Also, given our assumption that $\sigma$ is sign-preserving and $|\sigma(x)| \leq |x|$ (i.e., it does not increase the norm of its input), we show that $\mathcal{E}_{net} \leq \mathcal{E}_{lin}$:

$$\begin{aligned}\mathcal{E}_{net} &= \|f_t\|^2 + (K\nabla f, \sigma(K\nabla f)) \\ &\leq \|f_t\|^2 + (K\nabla f, K\nabla f) = \mathcal{E}_{lin}\end{aligned}$$

Therefore, we conclude that $\mathcal{E}_{net} \leq c_K$.

□

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

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

## B Datasets

We report the statistics of the datasets used in this paper in Tab. 2.

## C Architectures in details

In this section we elaborate on the specific architectures that were used in our experiments in Sec. 4. As discussed in Sec. 3.3, all our network architectures are comprised of an opening layer ($1 \times 1$ convolution), a sequence of PDE-GCN layers, and a closing layer ($1 \times 1$ convolution), and possibly additional final convolution steps which serve as the classifier. In total, we have two types of architectures in our experiments, which differ in their classifier layers. Throughout the following tables, $c_{in}$ and $c_{out}$ denote the input and output channels, respectively, and $c$ denotes the number of features in hidden layers. We denote the number of PDE-GCN blocks by $L$, and the dropout probability by $p$. We use the Adam [47] 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. Our code is implemented using PyTorch [48], trained on an Nvidia Titan RTX GPU.

Our first architecture is described in Tab. 3 and includes only a closing layer as a final step. The architecture is used for the fully supervised node classification tasks. Note, the high-level architecture is the same as in GCNII [19], and only differs in the employed GCN-block, which is our PDE-GCN.

Table 3: The architecture used for semi-and fully supervised node classification and inductive learning.

| Input size | Layer | Output size |
|---|---|---|
| $n \times c_{in}$ | $1 \times 1$ Dropout(p) | $n \times c_{in}$ |
| $n \times c_{in}$ | $1 \times 1$ Convolution | $n \times c$ |
| $n \times c$ | ReLU | $n \times c$ |
| $n \times c$ | $L\times$ PDE-GCN block | $n \times c$ |
| $n \times c$ | $1 \times 1$ Dropout(p) | $n \times c$ |
| $n \times c$ | $1 \times 1$ Convolution | $n \times c_{out}$ |

The second architecture is used for the dense-shape correspondence task on FAUST is given in Tab. 4. In addition to the closing $1 \times 1$ convolution layer, it also includes a layer of a $1 \times 1$ convolution and an $ELU$ activation, followed by another final $1 \times 1$ convolution which classifies the point-to-point correspondence. In the case of the FAUST dataset, each mesh has $n = 6890$ vertices.

## D Hyper-parameters details

We provide the selected hyper-parameters in our experiments. We denote the learning rate of our PDE-GCN layers by by $LR_{GCN}$, and the learning rate of the $1 \times 1$ opening and closing as well as any additional classifier layers by $LR_{oc}$. Also, the weight decay for the opening and closing layers is denoted by $WD_{oc}$. For the PDE-GCN layers, no weight decay is used throughout all experiments.

Table 4: The architecture used for dense-shape correspondence on FAUST.

| Input size | Layer | Output size |
|---|---|---|
| $n \times 4$ | $1 \times 1$ Convolution | $n \times c$ |
| $n \times c$ | ReLU | $n \times c$ |
| $n \times c$ | $L\times$ PDE-GCN block | $n \times c$ |
| $n \times c$ | $1 \times 1$ Convolution | $n \times c$ |
| $n \times c$ | ReLU | $n \times c$ |
| $n \times c$ | $1 \times 1$ Convolution | $n \times 512$ |
| $n \times 512$ | ELU | $n \times 512$ |
| $n \times 512$ | Fully-Connected | $n \times n$ |

## D.1 Learning PDE dynamics

In this experiment we used a 8 layers mixed PDE-GCN$_\mathrm{M}$, starting with $\alpha = 0.5$, such that it is balanced between a PDE-GCN$_\mathrm{D}$ and a PDE-GCN$_\mathrm{H}$. We report the hyper-parameters for this experiment in Tab. 5.

Table 5: Learning PDE dynamics hyper-parameters

| Dataset | $LR_{GCN}$ | $LR_{oc}$ | $LR_{\alpha}$ | $WD_{oc}$ | $\#Channels$ | $Dropout$ | $h$ |
|---|---|---|---|---|---|---|---|
| Cora | $1 \cdot 10^{-4}$ | 0.01 | 0.01 | $5 \cdot 10^{-4}$ | 64 | 0.6 | 0.5 |
| FAUST | 0.001 | 0.01 | 0.01 | 0 | 256 | 0 | 0.01 |

## D.2 Fully-supervised node-classification

The hyper-parameters for this experiment are summarized in Tab. 6.

Table 6: Fully-Supervised classification hyper-parameters

| Dataset | $LR_{GCN}$ | $LR_{oc}$ | $WD_{oc}$ | $\#Channels$ | $Dropout$ | $h$ |
|---|---|---|---|---|---|---|
| Cora | $4 \cdot 10^{-5}$ | 0.06 | $1 \cdot 10^{-4}$ | 64 | 0.6 | 0.65 |
| CiteSeer | $2 \cdot 10^{-4}$ | 0.07 | $1 \cdot 10^{-4}$ | 64 | 0.6 | 0.4 |
| PubMed | $5 \cdot 10^{-5}$ | 0.02 | $3 \cdot 10^{-4}$ | 64 | 0.5 | 0.55 |
| Chameleon | $40 \cdot 10^{-4}$ | 0.02 | $8 \cdot 10^{-5}$ | 64 | 0.6 | 0.55 |
| Cornell | $2.5 \cdot 10^{-4}$ | 0.07 | $2.5 \cdot 10^{-4}$ | 64 | 0.5 | 0.05 |
| Texas | $3 \cdot 10^{-4}$ | 0.05 | $1 \cdot 10^{-4}$ | 64 | 0.5 | 0.05 |
| Wisconsin | $3 \cdot 10^{-5}$ | 0.07 | $5 \cdot 10^{-5}$ | 64 | 0.5 | 0.054 |