DIRECTIONAL GRAPH NETWORKS

Dominique Beaini* **& Vincent Létourneau**
InVivo AI
Montreal, QC, H2S 3H1, Canada
{dominique,vincent}@invivoai.com

William L. Hamilton
McGill University, MILA
Montreal, QC, H3A 0G4
wlh@cs.mcgill.ca

Saro Passaro* **& Gabriele Corso & Pietro Liò**
department of Computer Science and Technology
University of Cambridge
Cambridge, United Kingdom
{sp976,gc579,pl219}@cam.ac.uk

ABSTRACT

The lack of anisotropic kernels in graph neural networks (GNNs) strongly limits their expressiveness, contributing to well-known issues such as over-smoothing. To overcome this limitation, we propose the first globally consistent anisotropic kernels for GNNs, allowing for graph convolutions that are defined according to topologicaly-derived directional flows. First, by defining a vector field in the graph, we develop a method of applying directional derivatives and smoothing by projecting node-specific messages into the field. Then, we propose the use of the Laplacian eigenvectors as such vector field. We show that the method generalizes CNNs on an n-dimensional grid and is provably more discriminative than standard GNNs regarding the Weisfeiler-Lehman 1-WL test. We evaluate our method on different standard benchmarks and see a relative error reduction of 8% on the CIFAR10 graph dataset and 11% to 32% on the molecular ZINC dataset, and a relative increase in precision of 1.6% on the MolPCBA dataset. An important outcome of this work is that it enables graph networks to embed directions in an unsupervised way, thus allowing a better representation of the anisotropic features in different physical or biological problems.

1 INTRODUCTION

One of the most important distinctions between convolutional neural networks (CNNs) and graph neural networks (GNNs) is that CNNs allow for any convolutional kernel, while most GNN methods are limited to symmetric kernels (also called isotropic kernels in the literature) Kipf & Welling (2016); Xu et al. (2018a); Gilmer et al. (2017). There are some implementation of asymmetric kernels using gated mechanisms Bresson & Laurent (2017); Veličković et al. (2017), motif attention Peng et al. (2019), edge features Gilmer et al. (2017) or by using the 3D structure of molecules for message passing Klicpera et al. (2019).

However, to the best of our knowledge, there are currently no methods that allow asymmetric graph kernels that are dependent on the full graph structure or on directional flows. They either depend on local structures or local features. This is in opposition to images which exhibit canonical directions: the horizontal and vertical axes.

We propose a novel idea for GNNs: use vector fields in the graph to define directions for the propagation of information. Hence, the aggregation or message passing will be projected onto these directions so that the contribution of each neighbouring node n_v will be weighted by its alignment with the vector fields at the receiving node n_u. This enables our method to propagate information via directional derivatives or smoothing of the features.

*equal contribution

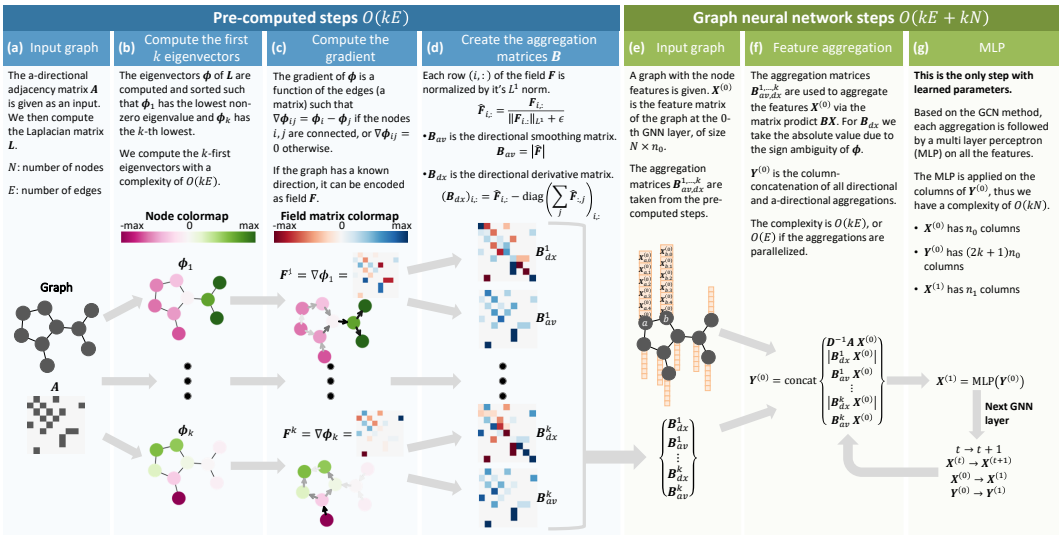


Figure 1: Overview of the steps required to aggregate messages in the direction of the eigenvectors.

We also explore using the gradients of the low-frequency eigenvectors of the Laplacian of the graph ϕ_k, since they exhibit interesting properties Bronstein et al. (2017); Chung et al. (1997). In particular, they can be used to define optimal partitions of the nodes in a graph, to give a natural ordering Levy (2006), and to find the dominant directions of the graph diffusion process Chung & Yau (2000). Further, we show that they generalize the horizontal and vertical directional flows in a grid, allowing them to guide the aggregation and mimic the asymmetric and directional kernels present in computer vision. In fact, we demonstrate mathematically that our work generalizes CNNs by reproducing all convolutional kernels of radius R in a n-dimensional grid.

We further show that our directional graph network (DGN) model theoretically and empirically allows for efficient message passing across distant communities, which reduces the well known problem of over-smoothing, and aligns well with the need of independent aggregation rules Corso et al. (2020). Finally, in Appendix E, we show how the underlying vector fields can be used to define data augmentation techniques on graphs.

2 Theoretical Development

2.1 Vector Fields in a Graph

Based on a recent review from Bronstein et al. (2017), this section presents the ideas of differential geometry applied to graphs, with the goal of finding proper definitions of scalar products, gradients and directional derivatives.

Let $G = (V, E)$ be a graph with V the set of vertices and $E \subset V \times V$ the set of edges. Define the vector spaces $L^2(V)$ and $L^2(E)$ as the set of maps $V \to \mathbb{R}$ and $E \to \mathbb{R}$ with $\boldsymbol{x}, \boldsymbol{y} \in L^2(V)$ and $\boldsymbol{F}, \boldsymbol{H} \in L^2(E)$ and scalar products

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle_{L^2(V)} := \sum_{i \in V} \boldsymbol{x}_i \boldsymbol{y}_i \quad , \qquad \langle \boldsymbol{F}, \boldsymbol{H} \rangle_{L^2(E)} := \sum_{(i,j) \in E} \boldsymbol{F}_{(i,j)} \boldsymbol{H}_{(i,j)} \tag{1}$$

Think of E as the "tangent space" to V and of $L^2(E)$ as the set of "vector fields" on the space V with each row $\boldsymbol{F}_{i,:}$ representing a vector at the i-th node. Define the pointwise scalar product as the map $L^2(E) \times L^2(E) \to L^2(V)$ taking 2 vector fields and returning their inner product at each point of V, at the node i is defined by equation 2.

$$\langle \boldsymbol{F}, \boldsymbol{H} \rangle_i := \sum_{j:(i,j) \in E} \boldsymbol{F}_{i,j} \boldsymbol{H}_{i,j} \tag{2}$$

In equation 3, we define the gradient ∇ as a mapping $L^2(V) \to L^2(E)$ and the divergence div as a mapping $L^2(E) \to L^2(V)$, thus leading to an analogue of the directional derivative in equation 4.

$$(\nabla \boldsymbol{x})_{(i,j)} := \boldsymbol{x}(j) - \boldsymbol{x}(i) \quad , \qquad (\text{div}\, \boldsymbol{F})_i := \sum_{j:(i,j)\in E} \boldsymbol{F}_{(i,j)} \tag{3}$$

Definition 1. *The directional derivative of the function $\boldsymbol{x}$ on the graph G in the direction of the vector field $\hat{\boldsymbol{F}}$ where each vector is of unit-norm is*

$$D_{\hat{\boldsymbol{F}}}\boldsymbol{x}(i) := \langle \nabla \boldsymbol{x}, \hat{\boldsymbol{F}} \rangle_i = \sum_{j:(i,j)\in E} (\boldsymbol{x}(j) - \boldsymbol{x}(i))\hat{\boldsymbol{F}}_{i,j} \tag{4}$$

2.2 Directional smoothing and derivatives

Next, we show how the vector field $\boldsymbol{F}$ is used to *guide* the graph aggregation by projecting the incoming messages. Specifically, we define the weighted aggregation matrices $\boldsymbol{B}_{av}$ and $\boldsymbol{B}_{dx}$ that allow to compute the directional smoothing and directional derivative of the node features.

The directional average matrix $\boldsymbol{B}_{av}$ is the weighted aggregation matrix such that all weights are positives and all rows have an L^1-norm equal to 1, as shown in equation 5 and theorem 2.1, with a proof in the appendix F.1.

$$\boldsymbol{B}_{av}(\boldsymbol{F})_{i,:} = \frac{|\boldsymbol{F}_{i,:}|}{||\boldsymbol{F}_{i,:}||_{L^1} + \epsilon} \tag{5}$$

The variable ϵ is an arbitrarily small positive number used to avoid floating point errors. The aggregator works by assigning a large weight to the elements in the forward or backward direction of the field, while assigning a small weight to the other elements, with a total weight of 1.

Theorem 2.1 (Directional smoothing)**.** *The operation $\boldsymbol{y} = \boldsymbol{B}_{av}\boldsymbol{x}$ is the directional average of $\boldsymbol{x}$, in the sense that $\boldsymbol{y}_u$ is the mean of $\boldsymbol{x}_v$, weighted by the direction and amplitude of $\boldsymbol{F}$.*

The directional derivative matrix $\boldsymbol{B}_{dx}$ is defined in (6) and theorem 2.2, (proof in appendix F.2).

$$\boldsymbol{B}_{dx}(\boldsymbol{F})_{i,:} = \hat{\boldsymbol{F}}_{i,:} - \text{diag}\Big(\sum_j \hat{\boldsymbol{F}}_{:,j}\Big)_{i,:} \qquad \hat{\boldsymbol{F}}_{i,:} = \left(\frac{\boldsymbol{F}_{i,:}}{||\boldsymbol{F}_{i,:}||_{L^1} + \epsilon}\right) \tag{6}$$

Theorem 2.2 (Directional derivative)**.** *Suppose $\hat{\boldsymbol{F}}$ have rows of unit L^1 norm. The operation $\boldsymbol{y} = \boldsymbol{B}_{dx}(\hat{\boldsymbol{F}})\boldsymbol{x}$ is the centered directional derivative of $\boldsymbol{x}$ in the direction of $\boldsymbol{F}$.*

These aggregators are directional, interpretable and complementary, making them ideal choices for GNNs. We discuss the choice of aggregators in more details in appendix B.

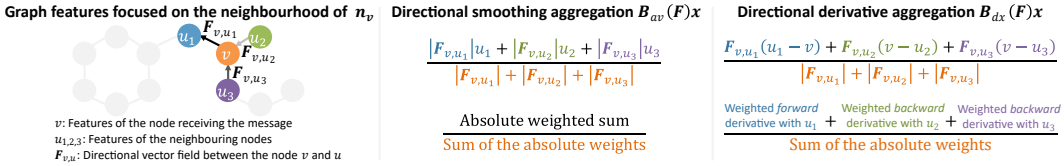


Figure 2: Illustration of how the directional aggregation works at a node n_v, with the arrows representing the direction and intensity of the field $\boldsymbol{F}$.

2.3 Gradient of the eigenvectors as interpretable vector fields

In this section we give theoretical support for the choice of gradients of the eigenfunctions of the Laplacian ($\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$) as sensible vectors along which to do directional message passing since they are interpretable and allow to reduce the over-smoothing.

The problems of *over-smoothing* and *over-squashing* are critical issues in GNNs Alon & Yahav (2020); Hamilton (2020). Over-smoothing results in representations tending to reach a mean-field

equilibrium Hamilton (2020). Over-squashing reflects the inability for GNNs to propagate informative signals between distant nodes Alon & Yahav (2020). Both problems are related to the fact that the influence of one node's input on the final representation of another node in a GNN is determined by the likelihood of the two nodes co-occurring on a truncated random walk Xu et al. (2018b). We show in theorem 2.3 (proved in appendix F.4) that by passing information in the direction of $\boldsymbol{\phi}_1$, the eigenvector associated to the lowest non-trivial frequency of $\boldsymbol{L}_{\text{norm}}$, DGNs can efficiently share information between the farthest nodes of the graph, when using the *K-walk distance* (definition F.3) to measure the difficulty of passing information. Thus, DGNs provide a natural way to address both the over-smoothing and over-squashing problems: they can efficiently propagate messages between distant nodes and in a direction that counteracts over-smoothing.

Theorem 2.3 (K-Gradient of low frequency eigenvectors)**.** *Let λ_i and $\boldsymbol{\phi}_i$ be the eigenvalues and eigenvectors of $\boldsymbol{L}_{norm}$ of a connected graph and $a, b = \arg\max_{1\leq i,j\leq n}\{d_K(v_i, v_j)\}$ be the nodes that have highest K-walk distance. Let $m = \arg\min_{1\leq i\leq n}(\boldsymbol{\phi}_1)_i$ and $M = \arg\max_{1\leq i\leq n}(\boldsymbol{\phi}_1)_i$, then v_m, v_M appr. v_a, v_b with order $O(\lambda_2 - \lambda_1)$.*

2.4 Generalization of the convolution on a grid

In this section we show that our method generalizes CNNs by allowing to define any radius-R convolutional kernels in grid-shaped graphs. The radius-R kernel at node u is a convolutional kernel that takes the weighted sum of all nodes v at a distance $d(u, v) \leq R$.

Consider the lattice graph Γ of size $N_1 \times ... \times N_n$ where each vertices are connected to their direct non-diagonal neighbour. We know from lemma F.2 that, for each dimension, there is an eigenvector that is only a function of this specific dimension. Hence, the Laplacian eigenvectors of the grid can play a role analogous to the axes in Euclidean space. With this knowledge, we show in theorem 2.4 (proven in F.10), that we can generalize all convolutional kernels in an n-dimensional grid.

Theorem 2.4 (Generalization radius-R convolutional kernel in a lattice)**.** *For an n-dimensional lattice, any convolutional kernel of radius R can be realized by a linear combination of directional aggregation matrices and their compositions.*

2.5 Comparison with Weisfeiler-Lehman (WL) test

We also compare the expressiveness of the Directional Graph Networks with the classical WL graph isomorphism test which is often used to classify the expressivity of graph neural networks (Xu et al., 2018a). In theorem 2.5 (proven in appendix F.11) we show that DGNs are capable of distinguishing pairs of graphs that the 1-WL test (and so ordinary GNNs) cannot differentiate.

Theorem 2.5 (Comparison with 1-WL test)**.** *DGNs using the mean aggregator, any directional aggregator of the first Laplacian eigenvector and injective degree-scalers are strictly more powerful than the 1-WL test.*

3 Results and discussion

We use a variety of benchmarks proposed by Dwivedi et al. (2020) to test the empirical performance of our proposed methods. We fine tuned it on the various datasets and we report its performance in figure 3. We observe that DGN provides significant improvement across all benchmarks, highlighting the importance of anisotropic kernels. Furthermore, in appendix D we provide a fair comparison between the *mean-aggregator* baseline and models using directional aggregators.

4 Conclusion

The proposed DGN method allows to solve many problems of GNNs, including the lack of anisotropy, the low expressiveness, the over-smoothing and over-squashing. For the first time in graph networks, we generalize the directional properties of CNNs and their data augmentation capabilities. Based on an intuitive idea and backed by a set of strong theoretical and empirical results, we believe this work will give rise to a new family of directional GNNs. Future work can focus on the implementation of radius-R kernels and improving the choice of multiple orthogonal directions.

Model	ZINC		PATTERN	CIFAR10		MolHIV	MolPCBA
	No edge features MAE	Edge features MAE	No edge features % acc	No edge features % acc	Edge features % acc	No edge features % ROC-AUC	All models % AP
GCN	$0.469_{\pm 0.002}$		$65.880_{\pm 0.074}$	$54.46_{\pm 0.10}$		$76.06_{\pm 0.97}$ *	$20.20_{\pm 0.24}$ *
GIN	$0.408_{\pm 0.008}$		$85.590_{\pm 0.011}$	$53.28_{\pm 3.70}$		$75.58_{\pm 1.40}$ *	$22.66_{\pm 0.28}$ *
GraphSage	$0.410_{\pm 0.005}$		$50.516_{\pm 0.001}$	$66.08_{\pm 0.24}$			
GAT	$0.463_{\pm 0.002}$		$75.824_{\pm 1.823}$	$65.48_{\pm 0.33}$			
MoNet	$0.407_{\pm 0.007}$		$85.482_{\pm 0.037}$	$53.42_{\pm 0.43}$			
GatedGCN	$0.422_{\pm 0.006}$	$0.363_{\pm 0.009}$	$84.480_{\pm 0.122}$	$69.19_{\pm 0.28}$	$69.37_{\pm 0.48}$		
PNA	$0.320_{\pm 0.032}$	$0.188_{\pm 0.004}$	$86.567_{\pm 0.075}$	$70.46_{\pm 0.44}$	$70.47_{\pm 0.72}$	$79.05_{\pm 1.32}$ *	$28.38_{\pm 0.35}$ *
DGN	$\mathbf{0.219_{\pm 0.010}}$	$\mathbf{0.168_{\pm 0.003}}$	$\mathbf{86.680_{\pm 0.034}}$	$\mathbf{72.70_{\pm 0.54}}$	$\mathbf{72.84_{\pm 0.42}}$	$\mathbf{79.70_{\pm 0.97}}$	$\mathbf{28.85_{\pm 0.30}}$ *

Figure 3: Fine tuned results of the DGN model against other models from Dwivedi et al. (2020) and Corso et al. (2020): GCN (Kipf & Welling, 2016), GIN (Xu et al., 2018a), GAT (Veličković et al., 2017), MoNet (Monti et al., 2017), GatedGCN (Bresson & Laurent, 2017) and PNA (Corso et al., 2020). Further details are provided in appendix D

REFERENCES

Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. *arXiv:2006.05205 [cs, stat]*, 2020. URL http://arxiv.org/abs/2006.05205.

Xavier Bresson and Thomas Laurent. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*, 2017.

Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34 (4):18–42, 2017. ISSN 1053-5888, 1558-0792. doi: 10.1109/MSP.2017.2693418. URL http://arxiv.org/abs/1611.08097.

Fan Chung and S.T.Yau. Discrete green's functions, 2000.

Fan Chung and S. T. Yau. Discrete green's functions. *Journal of Combinatorial Theory, Series A*, 91(1):191–214, 2000. ISSN 0097-3165. doi: 10.1006/jcta.2000.3094. URL http://www.sciencedirect.com/science/article/pii/S0097316500930942.

F.R.K. Chung, F.C. Graham, CBMS Conference on Recent Advances in Spectral Graph Theory, National Science Foundation (U.S.), American Mathematical Society, and Conference Board of the Mathematical Sciences. *Spectral Graph Theory*. CBMS Regional Conference Series. Conference Board of the mathematical sciences, 1997. ISBN 9780821803158. URL https://books.google.ca/books?id=4IK8DgAAQBAJ.

Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *arXiv preprint arXiv:2004.05718*, 2020.

Vishwaraj Doshi and Do Young Eun. Fiedler vector approximation via interacting random walks. *arXiv:2002.00283 [math]*, 2000. doi: 10.1145/3379487. URL http://arxiv.org/abs/2002.00283.

Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

Miroslav Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23:298–305, 01 1973. doi: 10.21136/CMJ.1973.101168.

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*, pp. 1263–1272. JMLR. org, 2017.

William L. Hamilton. *Graph Representation Learning*. Morgan and Claypool, 2020.

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*, pp. 770–778, 2016.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Md Amirul Islam, Sen Jia, and Neil D. B. Bruce. How much position information do convolutional neural networks encode? *arXiv:2001.08248 [cs]*, 2020. URL http://arxiv.org/abs/2001.08248.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. *arXiv:1802.04364 [cs, stat]*, 2018. URL http://arxiv.org/abs/1802.04364.

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

Johannes Klicpera, Janek Groß, and Stephan Günnemann. Directional message passing for molecular graphs. *ICLR2020*, 2019. URL https://openreview.net/forum?id=B1eWbxStPH.

Boris Knyazev, Graham W Taylor, and Mohamed Amer. Understanding attention and generalization in graph neural networks. In *Advances in Neural Information Processing Systems*, pp. 4204–4214, 2019.

Risi Kondor, Hy Truong Son, Horace Pan, Brandon Anderson, and Shubhendu Trivedi. Covariant compositional networks for learning graphs. *arXiv preprint arXiv:1801.02144*, 2018.

Alex Krizhevsky, 2009.

Cornelius Lanczos. *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*. United States Governm. Press Office Los Angeles, CA, 1950.

B. Levy. Laplace-beltrami eigenfunctions towards an algorithm that "understands" geometry. In *IEEE International Conference on Shape Modeling and Applications 2006 (SMI'06)*, pp. 13–13, 2006. doi: 10.1109/SMI.2006.21.

Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. *arXiv preprint arXiv:1812.09902*, 2018.

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*, pp. 5115–5124, 2017.

Hao Peng, Jianxin Li, Qiran Gong, Senzhang Wang, Yuanxing Ning, and Philip S. Yu. Graph convolutional neural networks via motif-based attention. *arXiv:1811.08270 [cs]*, 2019. URL http://arxiv.org/abs/1811.08270.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018a.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, pp. 5453–5462, 2018b.

A APPENDIX - INTUITIVE OVERVIEW

One of the biggest limitations of current GNN methods compared to CNNs is the inability to do message passing in a specific direction such as the horizontal one in a grid graph. In fact, it is difficult to define directions or coordinates based solely on the shape of the graph.

The lack of directions strongly limits the discriminative abilities of GNNs to understand local structures and simple feature transformations. Most GNNs are invariant to the permutation of the neighbours' features, so the nodes' received signal is not influenced by swapping the features of 2 neighbours. Therefore, several layers in a deep network will be employed to understand these simple changes instead of being used for higher level features, thus over-squashing the message sent between 2 distant nodes Alon & Yahav (2020).

In this work, one of the main contribution is the realisation that low frequency eigenvectors of the Laplacian can overcome this limitation by providing a variety of intuitive directional flows. As a first example, taking a grid-shaped graph of size $N \times M$ with $\frac{N}{2} < M < N$, we find that the eigenvector associated to the smallest non-zero eigenvalue increases in the direction of the width N and the second one increases in the direction of the height M. This property generalizes to n-dimensional grids and motivated the use of gradients of eigenvectors as preferred directions for general graphs.

We validated this intuition by looking at the flow of the gradient of the eigenvectors for a variety of graphs, as shown in figure 4. For example, in the Minnesota map, the first 3 non-constant eigenvectors produce logical directions, namely South/North, suburb/city, and West/East.

Another important contribution also noted in figure 4 is the ability to define any kind of direction based on a prior knowledge of the problem. Hence, instead of relying on eigenvectors to find directions in a map, we can simply use the cardinal directions or the rush-hour traffic flow.

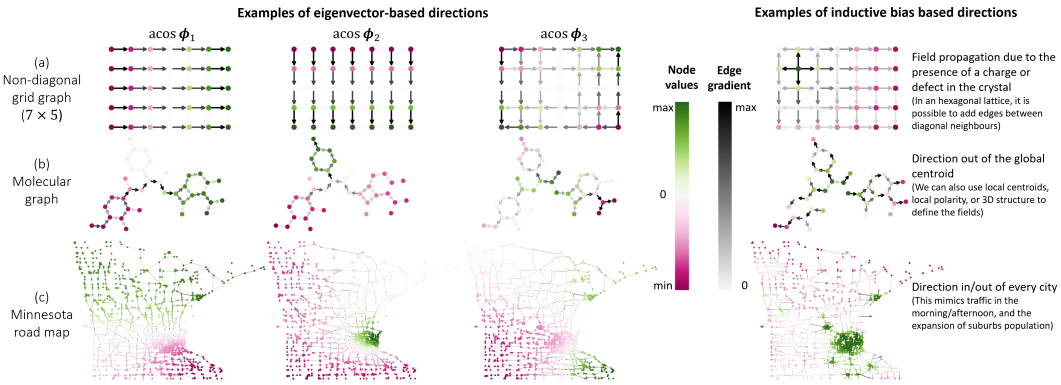


Figure 4: Possible directional flows in different types of graphs. The node coloring is a potential map and the edges represent the gradient of the potential with the arrows in the direction of the flow. The first 3 columns present the arcosine of the normalized eigenvectors (acos $\hat{\phi}$) as node coloring, and their gradients represented as edge intensity. The last column presents examples of inductive bias introduced in the choice of direction. (a) The eigenvectors 1 and 2 are the horizontal and vertical flows of the grid. (b) The eigenvectors 1 and 2 are the flow in the longest and second longest directions. (c) The eigenvectors 1, 2 and 3 flow respectively in the South-North, suburbs to city center and West-East directions. We ignore ϕ_0 since it is constant and has no direction.

B APPENDIX - CHOICES OF DIRECTIONAL AGGREGATORS

This appendix helps understand the choice of $\boldsymbol{B}_{av}$ and $\boldsymbol{B}_{dx}$ in section 2.2 and presents different directional aggregators that can be used as an alternative to the ones proposed.

A simple alternative to the directional smoothing and directional derivative operator is to simply take the *forward/backward* values according to the underlying positive/negative parts of the field $\boldsymbol{F}$, since it can effectively replicate them. However, there are many advantage of using $\boldsymbol{B}_{av,dx}$.

First, one can decide to use either of them and still have an interpretable aggregation with half the parameters. Then, we also notice that $\boldsymbol{B}_{av,dx}$ regularize the parameter by forcing the network to take both forward and backward neighbours into account at each time, and avoids one of the neighbours becoming too important. Lastly, they are robust to a change of sign of the eigenvectors since $\boldsymbol{B}_{av}$ is sign invariant and $\boldsymbol{B}_{dx}$ will only change the sign of the results, which is not the case for *forward/backward* aggregations.

B.1 Retrieving the mean and Laplacian aggregations

It is interesting to note that we can recover simple aggregators from the aggregation matrices $\boldsymbol{B}_{av}(\boldsymbol{F})$ and $\boldsymbol{B}_{dx}(\boldsymbol{F})$. Let $\boldsymbol{F}$ be a vector field such that all edges are equally weighted $\boldsymbol{F}_{ij} = \pm C$ for all edges (i, j). Then, the aggregator $\boldsymbol{B}_{av}$ is equivalent to a mean aggregation:

$$\boldsymbol{B}_{av}(\boldsymbol{F})\boldsymbol{x} = \boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{x}$$

Under the condition $F_{ij} = C$, the differential aggregator is equivalent to a Laplacian operator $\boldsymbol{L}$ normalized using the degree $\boldsymbol{D}$

$$\boldsymbol{B}_{dx}(C\boldsymbol{A})\boldsymbol{x} = \boldsymbol{D}^{-1}(\boldsymbol{A} - \boldsymbol{D})\boldsymbol{x} = -\boldsymbol{D}^{-1}\boldsymbol{L}\boldsymbol{x}$$

B.2 Global field normalization

The proposed aggregators are defined with a row-wise normalized field

$$\hat{\boldsymbol{F}_{i,:}} = \frac{\boldsymbol{F}_{i,:}}{||\boldsymbol{F}_{i,:}||_{L^P}}$$

meaning that all the vectors are of unit-norm and the aggregation/message passing is done only according to the direction of the vectors, not their amplitude. However, it is also possible to do a global normalization of the field $\boldsymbol{F}$ by taking a a matrix-norm instead of a vector-norm. Doing so will modulate the aggregation by the amplitude of the field at each node. One need to be careful since a global normalization might be very sensible to the number of nodes in the graph.

B.3 Center-balanced aggregators

A problem arises in the aggregators $\boldsymbol{B}_{dx}$ and $\boldsymbol{B}_{av}$ proposed in equations 5 and 6 when there is an imbalance between the positive and negative terms of $\boldsymbol{F}^{\pm}$. In that case, one of the directions overtakes the other in terms of associated weights.

An alternative is also to normalize the forward and backward directions separately, to avoid having either the backward or forward direction dominating the message.

$$\boldsymbol{B}_{av-center}(\boldsymbol{F})_{i,:} = \frac{\boldsymbol{F}'^{+}_{i,:} + \boldsymbol{F}'^{-}_{i,:}}{||\boldsymbol{F}'^{+}_{i,j} + \boldsymbol{F}'^{-}_{i,j}||_{L_1}} \quad , \qquad \boldsymbol{F}'^{\pm}_{i,:} = \frac{|\boldsymbol{F}^{\pm}_{i,:}|}{||\boldsymbol{F}^{\pm}_{i,:}||_{L^1} + \epsilon} \tag{7}$$

The same idea can be applied to the derivative aggregator equation 8 where the positive and negative parts of the field $\boldsymbol{F}^{\pm}$ are normalized separately to allow to project both the *forward* and *backward* messages into a vector field of unit-norm. $\boldsymbol{F}^{+}$ is the out-going field at each node and is used for the *forward* direction, while $\boldsymbol{F}^{-}$ is the in-going field used for the *backward* direction. By averaging the *forward* and *backward* derivatives, the proposed matrix $\boldsymbol{B}_{dx\text{-center}}$ represents the centered derivative matrix.

$$\boldsymbol{B}_{dx\text{-center}}(\boldsymbol{F})_{i,:} = \boldsymbol{F}'_{i,:} - \text{diag}\left(\sum_j \boldsymbol{F}'_{:,j}\right)_{i,:} \quad , \quad \boldsymbol{F}'_{i,:} = \frac{1}{2}\left(\underbrace{\frac{\boldsymbol{F}^{+}_{i,:}}{||\boldsymbol{F}^{+}_{i,:}||_{L^1} + \epsilon}}_{\text{forward field}} + \underbrace{\frac{\boldsymbol{F}^{-}_{i,:}}{||\boldsymbol{F}^{-}_{i,:}||_{L^1} + \epsilon}}_{\text{backward field}}\right) \tag{8}$$

B.4 HARDENING THE AGGREGATORS

The aggregation matrices that we proposed, mainly $\boldsymbol{B}_{dx}$ and $\boldsymbol{B}_{av}$ depend on a smooth vector field $\boldsymbol{F}$. At any given node, the aggregation will take a weighted sum of the neighbours in relation to the direction of $\boldsymbol{F}$. Hence, if the field $\boldsymbol{F}_v$ at a node v is *diagonal* in the sense that it gives a non-zero weight to many neighbours, then the aggregator will compute a weighted average of the neighbours.

Although there are clearly good reasons to have this weighted-average behaviour, it is not necessarily desired in every problem. For example, if we want to move a single node across the graph, this behaviour will smooth the node at every step. Instead, we propose below to soften and harden the aggregations by forcing the field into making a decision on the direction it takes.

Soft hardening the aggregation is possible by using a softmax with a temperature T on each row to obtain the field $\boldsymbol{F}_{\text{softhard}}$.

$$(\boldsymbol{F}_{\text{softhard}})_{i,:} = \text{sign}(\boldsymbol{F}_{i,:})\text{softmax}(T|\boldsymbol{F}_{i,:}|) \tag{9}$$

Hardening the aggregation is possible by using an infinite temperature, which changes the softmax functions into argmax. In this specific case, the node with the highest component of the field will be copied, while all other nodes will be ignored.

$$(\boldsymbol{F}_{\text{hard}})_{i,:} = \text{sign}(\boldsymbol{F}_{i,:})\text{argmax}(|\boldsymbol{F}_{i,:}|) \tag{10}$$

An alternative to the aggregators above is to take the *softmin/argmin* of the negative part and the *softmax/argmax* of the positive part.

B.5 FORWARD AND BACKWARD COPY

The aggregation matrices $\boldsymbol{B}_{av}$ and $\boldsymbol{B}_{dx}$ have the nice property that if the field is flipped (change of sign), the aggregation gives the same result, except for the sign of $\boldsymbol{B}_{dx}$. However, there are cases where we want to propagate information in the forward direction of the field, without smoothing it with the backward direction. In this case, we can define the strictly forward and strictly backward fields below, and use them directly with the aggregation matrices.

$$\boldsymbol{F}_{\text{forward}} = \boldsymbol{F}^{+} \quad , \qquad \boldsymbol{F}_{\text{backward}} = \boldsymbol{F}^{-} \tag{11}$$

Further, we can use the hardened fields in order to define a forward copy and backward copy, which will simply copy the node in the direction of the highest field component.

$$\boldsymbol{F}_{\text{forward copy}} = \boldsymbol{F}^{+}_{\text{hard}} \quad , \qquad \boldsymbol{F}_{\text{backward copy}} = \boldsymbol{F}^{-}_{\text{hard}} \tag{12}$$

B.6 PHANTOM ZERO-PADDING

Some recent work in computer vision have shown the importance of zero-padding to improve CNNs by allowing the network to understand it's position relative to the border Islam et al. (2020). In contrast, using boundary conditions or reflection padding makes the network completely blind to positional information. In this section, we show that we can mimic the zero-padding in the direction of the field $\boldsymbol{F}$ for both aggregation matrices $\boldsymbol{B}_{av}$ and $\boldsymbol{B}_{dx}$.

Starting with the $\boldsymbol{B}_{av}$ matrix, in the case of a missing neighbour in the forward/backward direction, the matrix will compensate by adding more weights to the other direction, due to the denominator which performs a normalization. Instead, we would need the matrix to consider both directions separately so that a missing direction would result in zero padding. Hence, we define $\boldsymbol{B}_{av,0pad}$ below, where either the F^{+} or F^{-} will be 0 on a boundary with strictly in-going/out-going field.

$$(\boldsymbol{B}_{av,0pad})_{i,:} = \frac{1}{2}\left(\frac{|\boldsymbol{F}^{+}_{i,:}|}{||\boldsymbol{F}^{+}_{i,:}||_{L^1} + \epsilon} + \frac{|\boldsymbol{F}^{-}_{i,:}|}{||\boldsymbol{F}^{-}_{i,:}||_{L^1} + \epsilon}\right) \tag{13}$$

Following the same argument, we define $\boldsymbol{B}_{dx,0pad}$ below, where either the forward or backward term are ignored. The diagonal term is also removed at the boundary so that the result is a center derivative equal to the subtraction of forward term with the 0-term on the back (or vice-versa), instead of a forward derivative.

$$\boldsymbol{B}_{dx-0pad}(\boldsymbol{F})_{i,:} = \begin{cases} \boldsymbol{F}'^{+}_{i,:} & \text{if } \sum_j \boldsymbol{F}'^{-}_{i,j} = 0 \\ \boldsymbol{F}'^{-}_{i,:} & \text{if } \sum_j \boldsymbol{F}'^{+}_{i,j} = 0 \\ \frac{1}{2}\left(\boldsymbol{F}'^{+}_{i,:} + \boldsymbol{F}'^{-}_{i,:} - \text{diag}\left(\sum_j \boldsymbol{F}'^{+}_{:,j} + \boldsymbol{F}'^{-}_{:,j}\right)_{i,:}\right), & \text{otherwise} \end{cases} \tag{14}$$

$$\boldsymbol{F}'^{+}_{i,:} = \frac{\boldsymbol{F}^{+}_{i,:}}{||\boldsymbol{F}^{+}_{i,:}||_{L^1} + \epsilon} \qquad \boldsymbol{F}'^{-}_{i,:} = \frac{\boldsymbol{F}^{-}_{i,:}}{||\boldsymbol{F}^{-}_{i,:}||_{L^1} + \epsilon}$$

B.7 Extending the radius of the aggregation kernel

Having aggregation kernels for neighbours of distance 2 or 3 is important to improve the expressiveness of GNNs, their ability to understand patterns, and to reduce the number of layers required. However, the lack of directions in GNNs strongly limits the radius of the kernels since, given a graph of regular degree d, a mean/sum aggregation at a radius-R will result in a heavy over-squashing of d^R messages. Using the directional fields, we can enumerate different paths, thus assigning a different weight for different R-distant neighbours. This method avoids the over-squashing, but empirical results are left for future work.

We aim at providing a general radius-R kernel $\boldsymbol{B}_R$ that assigns different weights to different subsets of nodes n_u at a distance R from the center node n_v.

First, we decompose the matrix $\boldsymbol{B}(\boldsymbol{F})$ into positive and negative parts $\boldsymbol{B}^{\pm}(\boldsymbol{F})$ representing the forward and backward steps aggregation in the field $\boldsymbol{F}$.

$$\boldsymbol{B}(\boldsymbol{F}) = \boldsymbol{B}^{+}(\boldsymbol{F}) - \boldsymbol{B}^{-}(\boldsymbol{F}) \tag{15}$$

Thus, defining $\boldsymbol{B}^{\pm}_{fb}(\boldsymbol{F})_{i,:} = \frac{\boldsymbol{F}^{\pm}_{i,:}}{||\boldsymbol{F}_{i,:}||_{L^p}}$, we can find different aggregation matrices by using different combinations of walks of radius R. First demonstrated for a grid in theorem 2.4, we generalize it in equation 16 for any graph G.

Definition 2 (General radius R n-directional kernel)**.** *Let S_n be the group of permutations over n elements with a set of directional fields $\boldsymbol{F}_i$.*

$$\boldsymbol{B}_R := \underbrace{\sum_{\substack{V=\{v_1,v_2,...,v_n\}\in\mathbb{N}^n \\ ||V||_{L^1}\leq R, \quad -R\leq v_i\leq R}}}_{\substack{\textit{Any choice of walk } V \textit{ with at most } R \textit{ steps} \\ \textit{using all combinations of } v_1, v_2, ..., v_n}} \underbrace{\sum_{\sigma\in S_n}}_{\substack{\textit{optional} \\ \textit{permutations}}} a_V \underbrace{\prod_{j=1}^{N} (\boldsymbol{B}^{sgn(v_{\sigma(j)})}_{fb}(\boldsymbol{F}_{\sigma(j)}))^{|v_{\sigma(j)}|}}_{\textit{Aggregator following the steps } V\textit{, permuted by } S_n} \tag{16}$$

In this equation, n is the number of directional fields and R is the desired radius. V represents all the choices of walk $\{v_1, v_2, ..., v_n\}$ in the direction of the fields $\{\boldsymbol{F}_1, \boldsymbol{F}_2, ..., \boldsymbol{F}_n\}$. For example, $V = \{3, 1, 0, -2\}$ has a radius $R = 6$, with 3 steps *forward* of $\boldsymbol{F}_1$, 1 step *forward* of $\boldsymbol{F}_2$, and 2 steps *backward* of $\boldsymbol{F}_4$. The sign of each $\boldsymbol{B}^{\pm}_{fb}$ is dependant to the sign of $v_{\sigma(j)}$, and the power $|v_{\sigma(j)}|$ is the number of aggregation steps in the directional field $\boldsymbol{F}_{\sigma(j)}$. The full equation is thus the combination of all possible choices of paths across the set of fields $\boldsymbol{F}_i$, with all possible permutations. Note that we are restricting the sum to v_i having only a possible sign; although matrices don't commute, we avoid choosing different signs since it will likely self-intersect a lower radius walk. The permutations σ are required since, for example, the path *up* $\rightarrow$ *left* is different (in a general graph) than the path *left* $\rightarrow$ *up*.

This matrix $\boldsymbol{B}_R$ has a total of $\sum_{r=0}^{R}(2n)^r = \frac{(2n)^{R+1}-1}{2n-1}$ parameters, with a high redundancy since some permutations might be very similar, e.g. for a grid graph we have that *up* $\rightarrow$ *left* is

identical to *left* $\rightarrow$ *up*. Hence, we can replace the permutation S_n by a reverse ordering, meaning that $\prod_j^N \boldsymbol{B}_j = \boldsymbol{B}_N ... \boldsymbol{B}_2 \boldsymbol{B}_1$. Doing so does not perfectly generalize the radius-R kernel for all graphs, but it generalizes it on a grid and significantly reduces the number of parameters to $\sum_{r=0}^{R} \sum_{l=1}^{min(n,r)} 2^r \binom{n}{l} \binom{r-1}{l-1}$.

C APPENDIX - IMPLEMENTATION DETAILS

C.1 BENCHMARKS AND DATASETS

We use a variety of benchmarks proposed by Dwivedi et al. (2020) and Hu et al. (2020) to test the empirical performance of our proposed methods. In particular, to have a wide variety of graphs and tasks we chose:

1. ZINC, a graph regression dataset from molecular chemistry. The task is to predict a score that is a subtraction of computed properties $logP - SA$, with $logP$ being the computed octanol-water partition coefficient, and SA being the synthetic accessibility score (Jin et al., 2018).
2. CIFAR10, a graph classification dataset from computer vision (Krizhevsky, 2009). The task is to classify the images into 10 different classes, with a total of 5000 training image per class and 1000 test image per class. Each image has 32×32 pixels, but the pixels have been clustered into a graph of ~ 100 super-pixels. Each super-pixel becomes a node in an *almost* grid-shaped graph, with 8 edges per node. The clustering uses the code from Knyazev et al. (2019), and results in a different number of super-pixels per graph.
3. PATTERN, a node classification synthetic benchmark generated with Stochastic Block Models, which are widely used to model communities in social networks. The task is to classify the nodes into 2 communities and it tests the fundamental ability of recognizing specific predetermined subgraphs.
4. MolHIV, a graph classification benchmark from molecular chemistry. The task is to predict whether a molecule inhibits HIV virus replication or not. The molecules in the training, validation and test sets are divided using a scaffold splitting procedure that splits the molecules based on their two-dimensional structural frameworks.
5. MolPCBA, a graph classification benchmark from molecular chemistry. It consists of measured biological activities of small molecules generated by high-throughput screening. The dataset consists of a total of 437,929 molecules divided using a scaffold slitting procedure and a set of 128 properties to predict for each.

For the results in figure 5, our goal is to provide a fair comparison to demonstrate the capacity of our proposed aggregators. Therefore, we compare the various methods on both types of architectures using the same hyperparameters tuned in previous works (Corso et al., 2020) for similar networks. The models vary exclusively in the aggregation method and the width of the architectures to keep a set parameter budget. Following the indication of the benchmarks' authors, we averaged the performances of the models on 4 runs with different initialization seeds for the benchmarks from Dwivedi et al. (2020) (ZINC, PATTERN and CIFAR10) and 10 runs for the ones from Hu et al. (2020) (MolHIV and MolPCBA[1]).

For the results in figure 3, we took the fine tuned results of other models from the corresponding public leaderboards by Dwivedi et al. (2020) and Hu et al. (2020). For the DGN results we fine tuned the model taking the lowest validation loss across runs with the following hyperparameters (you can also find the fine tuned commands in the documentation of the code repository):

1. ZINC: weight decay $\in \{1 \cdot 10^{-5}, 10^{-6}, 3 \cdot 10^{-7}\}$, aggregators $\in \{(mean, avg_1)$, $(mean, dx_1)$, $(mean, av_1, dx_1)$, $(mean, min, max, av_1)$, $(mean, min, max, dx_1)\}$
2. CIFAR10: weight decay $\in \{3 \cdot 10^{-6}\}$, dropout $\in \{0.1, 0.3\}$, aggregators $\in \{(mean, av_1, av_2)$, $(mean, dx_1, dx_2)$, $(mean, dx_1, dx_2, av_1, av_2)$, $(mean, max, min, dx_1, dx_2)$, $(mean, max, min, av_1, av_2)\}$

[1]For MolPCBA, due to the computational cost of running models in the large dataset and the relatively low variance, we only used 1 run for the results in figure 5, but 10 runs in those for figure 3

3. PATTERN: weight decay $\in \{0, 10^{-8}\}$, architecture $\in \{simple, complex\}$, aggregators $\in \{(mean, av_1), (mean, dx_1), (mean, av_1, dx_1)\}$
4. MolHIV: aggregators $\in \{(mean, dx_1), (mean, av_1), (mean, dx_1, av_1), (mean, max, dx_1), (mean, max, dx_1, av_1), (mean, max, min, av_1, dx_1)\}$, dropout $\in \{0.1, 0.3, 0.5\}$, L $\in \{4, 6\}$
5. for MolPCBA, given we did not start from any previously tuned architecture, we performed a line search with the following hyperparameters: mix of aggregators $\in \{mean, max, min, sum, dx_1, dx_2, av_1, av_2\}$, dropout $\in \{0.1, 0.2, 0.3, 0.4\}$, L $\in \{4, 6, 8\}$, weight decay $\in \{10^{-7}, 10^{-6}, 3 \cdot 10^{-6}, 10^{-5}, 3 \cdot 10^{-5}\}$, batch size $\in \{128.512.2048, 3072\}$, learning rate $\in \{10^{-2}, 10^{-3}, 5 \cdot 10^{-4}, 2 \cdot 10^{-4}\}$, learning rate patience $\in \{4, 6, 8\}$, learning rate reduce factor $\in \{0.5, 0.8\}$, architecture type $\in \{simple, complex, towers\}$, edge features dimension $\in \{0, 8, 16, 32\}$

In CIFAR10 it is impossible to numerically compute a deterministic vector field with eigenvectors due to the multiplicity of λ_1 being greater than 1. This is caused by the symmetry of the square image, and is extremely rare in real-world graphs. Therefore, we used as underlying vector field the gradient of the coordinates of the image. Note that these directions are provided in the nodes' features in the dataset and available to all models, that they are co-linear to the eigenvectors of the grid as per lemma F.2, and that they mimic the inductive bias in CNNs.

C.2 Implementation and computational complexity

Unlike several more expressive graph networks (Kondor et al., 2018; Maron et al., 2018), our method does not require a computational complexity superlinear with the size of the graph. The calculation of the first k eigenvectors during pretraining, done using Lanczos method (Lanczos, 1950) and the sparse module of Scipy, has a time complexity of $O(Ek)$ where E is the number of edges. During training the complexity is equivalent to a m-aggregator GNN $O(Em)$ (Corso et al., 2020) for the aggregation and $O(Nm)$ for the MLP.

To all the architectures we added residual connections (He et al., 2016), batch normalization (Ioffe & Szegedy, 2015) and graph size normalization (Dwivedi et al., 2020).

For some of the datasets with non-regular graphs, we combine the various aggregators with logarithmic degree-scalers as in Corso et al. (2020).

An important thing to note is that, for dynamic graphs, the eigenvectors need to be re-computed dynamically with the changing edges. Fortunately, there are random walk based algorithms that can estimate ϕ_1 quickly, especially for small changes to the graph (Doshi & Eun, 2000). In the current empirical results, we do not work with dynamic graphs.

C.3 Running time

The precomputation of the first four eigenvectors for all the graphs in the datasets takes $38s$ for ZINC, $96s$ for PATTERN and $120s$ for MolHIV on CPU. Table 1 shows the average running time on GPU for all the various model from figure 5. On average, the epoch running time is 15% slower for the DGN compared to the mean aggregation, but a faster convergence for DGN means that the total training time is on average 2% faster for DGN.

C.4 Eigenvector multiplicity

The possibility to define equivariant directions using the low-frequency Laplacian eigenvectors is subject to the uniqueness of those vectors. When the dimension of the eigenspaces associated with the lowest eigenvalues is 1, the eigenvectors are defined up to a constant factor. We propose the use of unit vector normalization and an absolute value to eliminate the scale and sign ambiguity. When the dimension of those eigenspaces is greater than 1, it is not possible to define equivariant directions using the eigenvectors.

Fortunately, it is very rare for the Laplacian matrix to have repeated eigenvalues in real-world datasets. We validate this claim by looking at ZINC and PATTERN datasets where we found no

Table 1: Average running time for the non-fine tuned models from figure 5. Each entry represents average time per epoch / average total training time. For the first four datasets, each of the models has a parameter budget $\sim 100k$ and was run on a Tesla T4 (15GB GPU). The *avg increase* row is the average of the relative running time of all rows compared to the *mean* row, with a negative value meaning a faster running time.

	ZINC			**PATTERN**	
Aggregators	**Simple**	**Complex**	**Complex-E**	**Simple**	**Complex**
mean	3.29s/1505s	3.58s/1584s	3.56s/1654s	153.1s/10154s	117.8s/9031s
mean dx_1	3.86s/1122s	3.77s/1278s	4.22s/1371s	144.9s/8109s	127.2s/8417s
mean dx_1 dx_2	4.23s/1360s	4.55s/1560s	4.63s/1680s	153.3s/8057s	167.9s/9326s
mean av_1	3.68s/1297s	3.84s/1398s	3.92s/1272s	128.0s/8680s	88.1s/7456s
mean av_1 av_2	3.95s/1432s	4.03s/1596s	4.07s/1721s	134.2s/8115s	170.4s/11114s
mean dx_1 av_1	3.89s/1079s	4.09s/1242s	4.58s/1510s	118.6s/6221s	144.2s/9112s
avg increase	+19%/-16%	+13%/-11%	+20%/-9%	-11%/-23%	+18%/+1%

	CIFAR10		**MolHIV**	**MolPCBA**	
Aggregators	**Simple**	**Complex**	**Simple**	**Complex**	**Complex-E**
mean	83.6s/10526s	78.7s/10900s	11.4s/2189s	279s/30128s	356s/38126s
mean dx_1			12.6s/2348s	304s/34129s	461s/43419s
mean dx_1 dx_2	98.4s/8405s	100.9s/5191s	14.1s/2345s	314s/36581s	334s/38363s
mean av_1			12.2s/2177s	297s/30316s	436s/54545s
mean av_1 av_2	117.1s/12834s	89.5s/14481s	13.9s/2150s	315s/42297s	333s/36641s
mean dx_1 av_1			14.0s/2070s	326s/37523s	461s/59109s
avg increase	+29%/+1%	+21%/-10%	+17%/+1%	+12%/+20%	+14%/+22%

graphs with repeated Fiedler vector and only one graph out of 26k with multiplicity of the second eigenvector greater than 1.

When facing a graph that presents repeated Laplacian eigenvalues, we propose to randomly shuffle, during training time, different eigenvectors randomly sampled in the eigenspace. This technique will act as a data augmentation of the graph during training time allowing the network to train with multiple directions at the same time.

D APPENDIX - RESULTS

In section 2.2 we provide a comparison between popular models and DGN. We fine-tuned the models with a parameter budget $\sim 100k$ using the low-frequency Laplacian eigenvectors to define the directions, except for CIFAR10 that uses the coordinates of the image. In ZINC we used aggregators $\{$*mean, dx_1, max, min*$\}$, in PATTERN $\{$*mean, dx_1, av_1*$\}$ and in CIFAR10 $\{$*mean, dx_1, dx_2, max*$\}$. Additionally we provide a fair comparison between the *mean-aggregator* baseline and models using directional aggregators as well as empirical results for the proposed augmentation.

D.1 DIRECTIONAL AGGREGATION

We present in figure 3 a fair comparison of various aggregation strategies using the same parameter budget and hyperparameters. We see a consistent boost in the performance for *simple*, *complex* and *complex with edges* models using directional aggregators compared to the *mean-aggregator* baseline. For brevity, we denote dx_i and av_i as the directional derivative $\boldsymbol{B}^i_{dx}$ and smoothing $\boldsymbol{B}^i_{av}$ aggregators.

In particular, we see a significant improvement on ZINC using the derivative aggregator. We believe this is due to the capacity to move efficiently messages across opposite parts of the molecule and to better understand the role of atom pairs. Further, the thesis that DGNs can bridge the gap between CNNs and GNNs is supported by the clear improvements on CIFAR10 over the baselines.

Aggregators	ZINC			PATTERN		CIFAR10		MolHIV	MolPCBA	
	Simple MAE	Complex MAE	Complex-E MAE	Simple % acc	Complex % acc	Simple % acc	Complex % acc	Simple % ROC-AUC	Complex % AP	Complex-E % AP
mean	0.316	0.353	0.262	80.77	83.34	55.9	62.8	75.1	26.04	26.38
mean pos_1	0.349	0.332	0.297	80.76	83.74			75.8	26.97	27.50
mean pos_1 pos_2	0.344	0.330	0.284	84.51	81.25			76.1	26.03	25.65
mean dx_1	0.296	**0.233**	**0.191**	84.22	83.44			78.0	26.79	**27.91**
mean dx_1 dx_2	0.337	0.271	0.205	81.61	86.62	52.9	**69.8**	76.5	**27.16**	26.55
mean av_1	0.317	0.332	0.276	84.54	83.21			78.4	25.97	26.66
mean av_1 av_2	0.367	0.332	0.260	85.12	85.38	**60.6**	65.1	77.1	25.61	26.67
mean dx_1 av_1	**0.290**	0.245	0.192	**85.17**	**86.68**			**79.0**	26.40	27.47

Best

Worst

Figure 5: Test set results using a parameter budget of 100k, with the same hyperparameters as Corso et al. (2020). The low-frequency Laplacian eigenvectors are used to define the directions, except for CIFAR10 that uses the coordinates of the image.

With our theoretical analysis in mind, we expected to perform well on PATTERN since the flow of the first eigenvectors are meaningful directions in a stochastic block model and passing messages using those directions allows the network to efficiently detect the two communities. The results match our expectations, outperforming all the previous models.

E APPENDIX - DATA AUGMENTATION

E.1 DEFINITION

Another important result is that the directions in the graph allow to replicate some of the most common data augmentation techniques used in computer vision, namely reflection and rotation. The main difference is that, instead of modifying the image, the proposed transformation is applied on the vector field defining the aggregation kernel.

Definition 3 (Reflection of the vector field)**.** *For a vector field $\boldsymbol{F}$, the reflected field is $-\boldsymbol{F}$.*

Let $\boldsymbol{F}_1, \boldsymbol{F}_2$ be vector field in a graph. Let α be the angle between the vector fields such that $\langle \boldsymbol{F}_1, \boldsymbol{F}_2 \rangle = \|\boldsymbol{F}_1\| \|\boldsymbol{F}_2\| \cos(\alpha)$. The vector field $\boldsymbol{F}_2^{\perp}$ is the component of $\boldsymbol{F}_2$ perpendicular to $\boldsymbol{F}_1$:

$$(\boldsymbol{F}_2^{\perp})_{i,:} = \frac{(\boldsymbol{F}_2 - \langle \boldsymbol{F}_1, \boldsymbol{F}_2 \rangle \boldsymbol{F}_1)_{i,:}}{||\boldsymbol{F}_2 - \langle \boldsymbol{F}_1, \boldsymbol{F}_2 \rangle \boldsymbol{F}_1||_{i,:}} \tag{17}$$

Definition 4 (Rotation of the vector fields)**.** *For $\boldsymbol{F}_1$ and $\boldsymbol{F}_2$ non-colinear vector fields, their rotation by the angle θ in the plane formed by $\{\boldsymbol{F}_1, \boldsymbol{F}_2\}$ is*

$$\boldsymbol{F}_1' = \boldsymbol{F}_1 \cos\theta + \boldsymbol{F}_2^{\perp} \sin\theta \quad , \qquad \boldsymbol{F}_2' = \boldsymbol{F}_1 \cos(\theta + \alpha) + \boldsymbol{F}_2^{\perp} \sin(\theta + \alpha) \tag{18}$$

E.2 RESULTS

To evaluate the effectiveness of the proposed augmentation, we trained the models on a reduced version of the CIFAR10 dataset. The results in figure 6 show clearly a higher expressive power of the *dx* aggregator, enabling it to fit well the training data. For a small dataset, this comes at the cost of overfitting and reduced performance on the test set. However, we observe that randomly rotating the kernels counteract the overfitting and allows the model to better generalize.

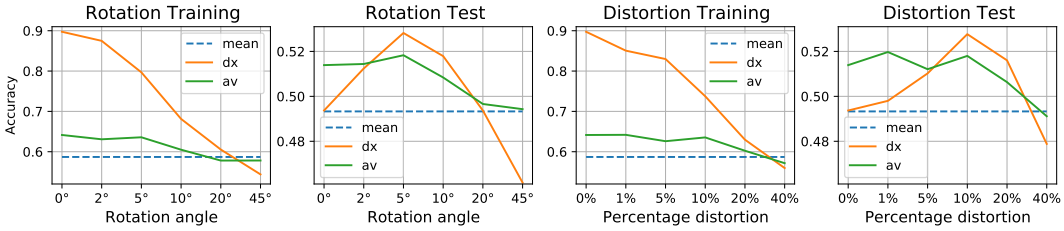


Figure 6: Accuracy of the various models using the *complex* architecture and trained on 10% of the CIFAR10 training set (4.5k images) with various ranges of rotation angles. An angle of x corresponds to a rotation of the kernel of a random angle sampled uniformly in $(-x^\circ, x^\circ)$ using definition 4 with $\boldsymbol{F}_{1,2}$ being the gradient of the horizontal/vertical coordinates. The *mean* baseline model is not affected by the augmentation since it does not use the underlining vector field.

F Appendix - Mathematical Proofs

F.1 Proof for Theorem 2.1 (Directional smoothing)

The operation $\boldsymbol{y} = \boldsymbol{B}_{av}\boldsymbol{x}$ is the directional average of $\boldsymbol{x}$, in the sense that $\boldsymbol{y}_u$ is the mean of $\boldsymbol{x}_v$, weighted by the direction and amplitude of $\boldsymbol{F}$.

Proof. This should be a simple proof, that if we want a weighted average of our neighbours, we simply need to multiply the weights by each neighbour, and divide by the sum of the weights. Of course, the weights should be positive.

□

F.2 Proof for Theorem 2.2 (Directional derivative)

Suppose $\hat{\boldsymbol{F}}$ have rows of unit L^1 norm. The operation $\boldsymbol{y} = \boldsymbol{B}_{dx}(\hat{\boldsymbol{F}})\boldsymbol{x}$ is the centered directional derivative of $\boldsymbol{x}$ in the direction of $\boldsymbol{F}$, in the sense of equation 4, i.e.

$$\boldsymbol{y} = D_{\hat{\boldsymbol{F}}}\boldsymbol{x} = \Big(\hat{\boldsymbol{F}} - \operatorname{diag}\Big(\sum_j \hat{\boldsymbol{F}}_{:,j}\Big)\Big)\boldsymbol{x}$$

Proof. Since $\boldsymbol{F}$ rows have unit L^1 norm, $\hat{\boldsymbol{F}} = \boldsymbol{F}$. The i-th coordinate of the vector $\left(\boldsymbol{F} - \operatorname{diag}\left(\sum_j \boldsymbol{F}_{:,j}\right)\right)\boldsymbol{x}$ is

$$\begin{aligned}\left(\boldsymbol{F}\boldsymbol{x} - \operatorname{diag}\left(\sum_j \boldsymbol{F}\right)\boldsymbol{x}\right)_i &= \sum_j \boldsymbol{F}_{i,j}\boldsymbol{x}(j) - \left(\sum_j \boldsymbol{F}_{i,j}\right)\boldsymbol{x}(i) \\ &= \sum_{j:(i,j)\in E} (\boldsymbol{x}(j) - \boldsymbol{x}(i))\boldsymbol{F}_{i,j} \\ &= D_{\boldsymbol{F}}\,\boldsymbol{x}(i)\end{aligned}$$

□

F.3 Definition of K-walk distance

Definition 5 (K-walk distance)**.** *The K-walk distance $d_K(v_i, v_j)$ on a graph is the average number of times v_i is hit in a K step random walk starting from v_j.*

F.4 PROOF FOR THEOREM 2.3 (K-GRADIENT OF LOW FREQUENCY EIGENVECTORS)

Let λ_i and $\boldsymbol{\phi}_i$ be the eigenvalues and eigenvectors of $\boldsymbol{L}_{norm}$ of a connected graph and $a, b = \arg\max_{1\leq i,j\leq n}\{d_K(v_i, v_j)\}$ be the nodes that have highest K-walk distance. Let $m = \arg\min_{1\leq i\leq n}(\boldsymbol{\phi}_1)_i$ and $M = \arg\max_{1\leq i\leq n}(\boldsymbol{\phi}_1)_i$, then v_m, v_M appr. v_a, v_b with order $O(\lambda_2 - \lambda_1)$.

Proof. First we need the following proposition:

Proposition 1 (K-walk distance matrix)**.** *The K-walk distance matrix $\boldsymbol{P}$ associated with a graph is the matrix such that $(\boldsymbol{P})_{i,j} = d_K(v_i, v_j)$ can be written as $\sum_{p=1}^{K} \boldsymbol{W}^p$, where $\boldsymbol{W} = \boldsymbol{D}^{-1}\boldsymbol{A}$ is the random walk matrix.*

Let's define $\boldsymbol{W} = \boldsymbol{D}^{-1}\boldsymbol{A}$ the random walk matrix of the graph.

First, we are going to show that W is jointly diagonalizable with $\boldsymbol{L}_{\text{norm}}$ and we are going to relate its eigenvectors $\boldsymbol{\phi}'_i$ and its eigenvalues λ'_i with the ones of $\boldsymbol{W}$.

Indeed, $\boldsymbol{L}_{\text{sym}}$ is a symmetric real matrix which is semi-positive definite diagonalizable by the spectral theorem. Since the matrix $\boldsymbol{L}_{\text{norm}}$ is similar to $\boldsymbol{D}^{\frac{1}{2}}\boldsymbol{L}_{\text{norm}}\boldsymbol{D}^{-\frac{1}{2}} = D^{-\frac{1}{2}}\boldsymbol{L}\boldsymbol{D}^{-\frac{1}{2}} = \boldsymbol{L}_{\text{sym}}$ and the matrix of similarity is $\boldsymbol{D}^{\frac{1}{2}}$, a positive definite matrix, $\boldsymbol{L}_{\text{norm}}$ is diagonalizable and semi-positive definite.

By

$$\boldsymbol{L}_{\text{norm}} = \boldsymbol{D}^{-1}\boldsymbol{L} = \boldsymbol{D}^{-1}(\boldsymbol{L} + \boldsymbol{D} - \boldsymbol{D}) = \boldsymbol{I} + \boldsymbol{D}^{-1}(\boldsymbol{L} - \boldsymbol{D}) = \boldsymbol{I} - \boldsymbol{D}^{-1}\boldsymbol{A} = \boldsymbol{I} - \boldsymbol{W}$$

the random walk matrix is jointly diagonalizable with the random walk Laplacian. Also their eigenvalues and eigenvectors are related to each other by $\boldsymbol{\phi}_i = \boldsymbol{\phi}'_{n-1-i}$ and $\lambda'_i = 1 - \lambda_{n-1-i}$

Moreover, the constant eigenvector associated with eigenvalue 0 of the Random walk Laplacian, is the eigenvector associated with the highest eigenvalue of the Random walk matrix and by the formula obtained, $\lambda'_{n-1} = 1 - \lambda_0 = 1$

Now, we are going to approximate the K-walk distance matrix $\boldsymbol{P}$ using the 2 eigenvectors of the Random walk matrix associated with the highest eigenvalues.

By Proposition 1 we have that $\boldsymbol{P} = \sum_{p=1}^{K} \boldsymbol{W}^p$, which can be written as

$$\sum_{p=1}^{K}(\sum_{i=0}^{n-1} \boldsymbol{\phi}'_i\boldsymbol{\phi}'^T_i(\lambda'_i))^p = \sum_{p=1}^{K}\sum_{i=0}^{n-1} \boldsymbol{\phi}'_i\boldsymbol{\phi}'^T_i(\lambda'_i)^p$$

by eigen-decomposition.

Since $\lambda_{n-1-i} = 1 - \lambda'_i$ and $\lambda_2 \gg \lambda_1$, we have that $\lambda'_{n-2} \gg \lambda'_{n-3}$, hence we can approximate

$$P = \sum_{p=1}^{K}(\sum_{i=0}^{n-1} \boldsymbol{\phi}'_i\boldsymbol{\phi}'_i(\lambda'_i)^p) \approx \sum_{p=1}^{K}(\sum_{i=n-2}^{n-1} \boldsymbol{\phi}'_i\boldsymbol{\phi}'^T_i(\lambda'_i)^p) + O(\lambda'_{n-2} - \lambda'_{n-3}) =$$

$$= \sum_{p=1}^{K}(\sum_{i=0}^{1} \boldsymbol{\phi}_i\boldsymbol{\phi}^T_i(1-\lambda_i)^p) + O(\lambda_2 - \lambda_1) = \sum_{p=1}^{K}(\boldsymbol{\phi}_0\boldsymbol{\phi}^T_0 + \boldsymbol{\phi}_1\boldsymbol{\phi}^T_1(1-\lambda_1)^p) + O(\lambda_2 - \lambda_1) = K\boldsymbol{\phi}_0\boldsymbol{\phi}^T_0 + \kappa\boldsymbol{\phi}_1\boldsymbol{\phi}^T_1 + O(\lambda_2 - \lambda_1)$$

where $\kappa = \sum_{p=1}^{K}(1 - \lambda_1)^p$ is a positive constant.

Now we are going to show that the farthest nodes with respect to the K-walk distance are the ones associated with the highest and lowest value of $\boldsymbol{\phi}_1$.

Indeed if we want to choose i, j to be at the farthest distance we need to minimise

$$(\boldsymbol{P})_{i,j} = (K\boldsymbol{\phi}_0\boldsymbol{\phi}^T_0 + \kappa\boldsymbol{\phi}_1\boldsymbol{\phi}^T_1)_{i,j} = \frac{K}{n} + \kappa\boldsymbol{\phi}_1(i)\boldsymbol{\phi}_1(j)$$

which is minimum when $\boldsymbol{\phi}_1(i)\boldsymbol{\phi}_1(j)$ is minimum.

The quantity $\boldsymbol{\phi}_1(i)\boldsymbol{\phi}_1(j)$ is minimised when it has negative sign and highest absolute value, hence when i, j are associated with the negative and positive values with the highest absolute value: the lowest and the highest value of $\boldsymbol{\phi}_1$.

□

F.5 INFORMAL ARGUMENT IN SUPPORT OF CONJECTURE F.1

Conjecture F.1. *Define the hitting time $Q(x, y)$ defined as the expected number of steps in a random walk starting from node x ending in node y with the probability transition $P(x, y) = \frac{1}{d_x}$. Suppose that x, y are uniformly distributed random nodes such that $\boldsymbol{\phi}_i(x) < \boldsymbol{\phi}_i(y)$. Let z be the node obtained from x by taking one step in the direction of $\nabla\boldsymbol{\phi}_i$, then the expected hitting time is decreased proportionally to λ_i^{-1} and*

$$\mathbb{E}_{x,y}[Q(z, y)] \leq \mathbb{E}_{x,y}[Q(x, y)]$$

Define the *hitting time* $Q(x, y)$ defined as the expected number of steps in a random walk starting from node x ending in node y with the probability transition $P(x, y) = \frac{1}{d_x}$. Suppose that x, y are uniformly distributed random nodes such that $\boldsymbol{\phi}_i(x) < \boldsymbol{\phi}_i(y)$. Let z be the node obtained from x by taking one step in the direction of $\nabla\boldsymbol{\phi}_i$, then the expected hitting time is decreased proportionally to λ_i^{-1} and

$$\mathbb{E}_{x,y}[Q(z, y)] \leq \mathbb{E}_{x,y}[Q(x, y)]$$

In Chung & S.T.Yau (2000), it is shown the hitting time $Q(x, y)$ is given by the equation

$$Q(x, y) = vol\left(\frac{\boldsymbol{G}(y, y)}{d_y} - \frac{\boldsymbol{G}(x, y)}{d_x}\right)$$

With λ_k and $\boldsymbol{\phi}_k$ being the k-th eigenvalues and eigenvectors of the symmetric normalized Laplacian L_{sym}, vol the sum of the degrees of all nodes, d_x the degree of node x and $\boldsymbol{G}$ Green's function for the graph

$$\boldsymbol{G}(x, y) = d_x^{\frac{1}{2}} d_y^{\frac{-1}{2}} \sum_{k>0} \frac{1}{\lambda_k} \boldsymbol{\phi}_k(x)\boldsymbol{\phi}_k(y)$$

Since the sign of the eigenvector is not deterministic, the choice $\boldsymbol{\phi}_i(x) < \boldsymbol{\phi}_i(y)$ is used to simplify the argument without having to consider the change in sign.

Supposing $\lambda_1 \ll \lambda_2$, the first term of the sum of $\boldsymbol{G}$ has much more weight than the following terms. With z obtained from x by taking a step in the direction of the gradient of $\boldsymbol{\phi}_1$ we have

$$\boldsymbol{\phi}_1(z) - \boldsymbol{\phi}_1(x) > 0$$

We want to show that the following inequality holds

$$\mathbb{E}_{x,y}(Q(z, y)) < \mathbb{E}_{x,y}(Q(x, y))$$

this is equivalent to the following inequality

$$\mathbb{E}_{x,y}[\boldsymbol{G}(z, y)] > \mathbb{E}_{x,y}[\boldsymbol{G}(x, y)]$$

By the hypothesis $\lambda_1 \ll \lambda_2$, we can approximate $\boldsymbol{G}(x, y) \sim d_x^{\frac{1}{2}} d_y^{\frac{-1}{2}} \frac{1}{\lambda_1} \boldsymbol{\phi}_1(x)\boldsymbol{\phi}_1(y)$ so the last inequality is equivalent to

$$\mathbb{E}_{x,y}\left[d_z^{\frac{1}{2}} d_y^{\frac{-1}{2}} \frac{1}{\lambda_1} \boldsymbol{\phi}_1(z)\boldsymbol{\phi}_1(y)\right] > \mathbb{E}_{x,y}\left[d_x^{\frac{1}{2}} d_y^{\frac{-1}{2}} \frac{1}{\lambda_1} \boldsymbol{\phi}_1(x)\boldsymbol{\phi}_1(y)\right]$$

Removing all equal terms from both sides, the inequality is equivalent to

$$\mathbb{E}_{x,y}\left[d_z^{\frac{1}{2}} \boldsymbol{\phi}_1(z)\right] > \mathbb{E}_{x,y}\left[d_x^{\frac{1}{2}} \boldsymbol{\phi}_1(x)\right]$$

But showing this last inequality is not easy. We know that $\boldsymbol{\phi}_1(z) > \boldsymbol{\phi}_1(x)$ and from the choice of z being a step in the direction of $\nabla\boldsymbol{\phi}_1$, we know it is less likely to be on the border of the graph so we believe $\mathbb{E}(d_z) \geq \mathbb{E}(d_x)$. Thus we also believe that the conjecture should hold in general.

We believe this should be true even without the assumption on λ_1 and λ_2 and for more eigenvectors than $\boldsymbol{\phi}_1$.

F.6 COROLLARY - REDUCES OVER-SQUASHING

Following the direction of $\nabla\boldsymbol{\phi}_1$ is an efficient way of passing information between the farthest nodes of the graph (in terms of the K-walk distance).

F.7 COROLLARY - REDUCES OVER-SMOOTHING

Following the direction of $\nabla\boldsymbol{\phi}_1$ allows the influence distribution between node representations to be decorrelated from random-walk hitting times (assuming the definition of influence introduced in Xu et al. (2018b)).

F.8 PROOF FOR LEMMA F.2 (COSINE EIGENVECTORS)

Consider the lattice graph Γ of size $N_1 \times N_2 \times ... \times N_n$, that has vertices $\prod_{i=1,...,n}\{1, ..., N_i\}$ and the vertices $(x_i)_{i=1,...,n}$ and $(y_i)_{i=1,...,n}$ are connected by an edge iff $|x_i - y_i| = 1$ for one index i and 0 for all other indices. Note that there are no diagonal edges in the lattice. The eigenvector of the Laplacian of the grid $L(\Gamma)$ are given by $\boldsymbol{\phi}_j$.

Lemma F.2 (Cosine eigenvectors)**.** *The Laplacian of Γ has an eigenvalue $2 - 2\cos\left(\frac{\pi}{N_i}\right)$ with the associated eigenvector $\boldsymbol{\phi}_j$ that depends only the variable in the i-th dimension and is constant in all others, with $\boldsymbol{\phi}_j = \mathbf{1}_{N_1} \otimes \mathbf{1}_{N_2} \otimes ... \otimes \boldsymbol{x}_{1,N_i} \otimes ... \otimes \mathbf{1}_{N_n}$, and $\boldsymbol{x}_{1,N_i}(j) = \cos\left(\frac{\pi j}{n} - \frac{\pi}{2n}\right)$*

Proof. First, recall the well known result that the path graph on N vertices P_N has eigenvalues

$$\lambda_k = 2 - 2\cos\left(\frac{\pi k}{n}\right)$$

with associated eigenvector $\boldsymbol{x}_k$ with i-th coordinate

$$\boldsymbol{x}_k(i) = \cos\left(\frac{\pi k i}{n} + \frac{\pi k}{2n}\right)$$

The Cartesian product of two graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$ is defined as $G \times H = (V_{G\times H}, E_{G\times H})$ with $V_{G\times H} = V_G \times V_H$ and $((u_1, u_2), ((v_1, v_2)) \in E_{G\times H}$ iff either $u_1 = v_1$ and $(u_2, v_2) \in E_H$ or $(u_1, v_1) \in V_G$ and $u_2 = v_2$. It is shown in Fiedler (1973) that if $(\mu_i)_{i=1,...,m}$ and $(\lambda_j)_{j=1,...,n}$ are the eigenvalues of G and H respectively, then the eigenvalues of the Cartesian product graph $G \times H$ are $\mu_i + \lambda_j$ for all possible eigenvalues μ_i and λ_j. Also, the eigenvectors associated to the eigenvalue $\mu_i + \lambda_j$ are $u_i \otimes v_j$ with u_i an eigenvector of the Laplacian of G associated to the eigenvalue μ_i and v_j an eigenvector of the Laplacian of H associated to the eigenvalue λ_j.

Finally, noticing that a lattice of shape $N_1 \times N_2 \times ... \times N_n$ is really the Cartesian product of path graphs of length N_1 up to N_n, we conclude that there are eigenvalues $2 - 2\cos\left(\frac{\pi}{N_i}\right)$. Denoting by $\mathbf{1}_{N_j}$ the vector in $\boldsymbol{R}^{N_j}$ with only ones as coordinates, then the eigenvector associated to the eigenvalue $2 - 2\cos\left(\frac{\pi}{N_i}\right)$ is

$$\mathbf{1}_{N_1} \otimes \mathbf{1}_{N_2} \otimes ... \otimes \boldsymbol{x}_{1,N_i} \otimes ... \otimes \mathbf{1}_{N_n}$$

where $\boldsymbol{x}_{1,N_i}$ is the eigenvector of the Laplacian of P_{N_i} associated to its first non-zero eigenvalue. $2 - 2\cos\left(\frac{\pi}{N_i}\right)$. □

F.9 RADIUS 1 CONVOLUTION KERNELS IN A GRID

In this section we show any radius 1 convolution kernel can be obtained as a linear combination of the $\boldsymbol{B}_{dx}(\nabla\boldsymbol{\phi}_i)$ and $\boldsymbol{B}_{av}(\nabla\boldsymbol{\phi}_i)$ matrices for the right choice of Laplacian eigenvectors $\boldsymbol{\phi}_i$. First we show this can be done for 1-d convolution kernels.

Theorem F.3. *On a path graph, any 1D convolution kernel of size 3 k is a linear combination of the aggregators $\boldsymbol{B}_{av}, \boldsymbol{B}_{dx}$ and the identity $\boldsymbol{I}$.*

Proof. Recall from the previous proof that the first non zero eigenvalue of the path graph P_N has associated eigenvector $\boldsymbol{\phi}_1(i) = \cos(\frac{\pi i}{N} - \frac{\pi}{2N})$. Since this is a monotone decreasing function in i, the i-th row of $\nabla \boldsymbol{\phi}_1$ will be

$$(0, ..., 0, s_{i-1}, 0, -s_{i+1}, 0, ..., 0)$$

with s_{i-1} and $s_{i+1} > 0$. We are trying to solve

$$(a\boldsymbol{B}_{av} + b\boldsymbol{B}_{dx} + c\mathbf{Id})_{i,:} = (0, ..., 0, x, y, z, 0, ..., 0)$$

with x, y, z, in positions $i-1, i$ and $i+1$. This simplifies to solving

$$a\frac{1}{\|s\|_{L^1}}|s| + b\frac{1}{\|s\|_{L^2}}s + c(0, 1, 0) = (x, y, z)$$

with $s = (s_{i-1}, 0, -s_{i+1})$, which always has a solution because $s_{i-1}, s_{i+1} > 0$. □

Theorem F.4 (Generalization radius-1 convolutional kernel in a grid)**.** *Let Γ be the n-dimensional lattice as above and let $\boldsymbol{\phi}_j$ be the eigenvectors of the Laplacian of the lattice as in theorem F.2. Then any radius 1 kernel k on Γ is a linear combination of the aggregators $\boldsymbol{B}_{av}(\boldsymbol{\phi}_i), \boldsymbol{B}_{dx}(\boldsymbol{\phi}_i)$ and $\boldsymbol{I}$.*

Proof. This is a direct consequence of F.3 obtained by adding n 1-dimensional kernels, with each kernel being in a different axis of the grid as per Lemma F.2. See figure 7 for a visual example in 2D.

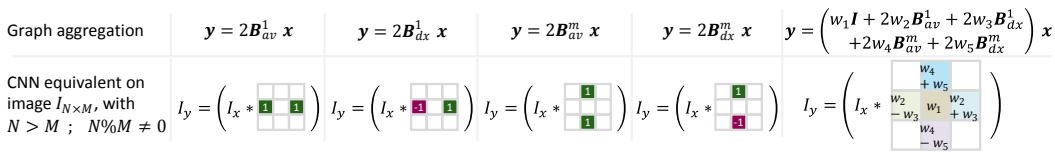


Figure 7: Realization of a radius-1 convolution using the proposed aggregators. I_x is the input feature map, $*$ the convolutional operator, I_y the convolution result, and $\boldsymbol{B}^i = \boldsymbol{B}(\nabla \boldsymbol{\phi}_i)$.

□

F.10 Proof for Theorem 2.4 (Generalization radius-R convolutional kernel in a lattice)

For an n-dimensional lattice, any convolutional kernel of radius R can be realized by a linear combination of directional aggregation matrices and their compositions.

Proof. For clarity, we first do the 2 dimensional case for a radius 2, then extended to the general case. Let k be the radius 2 kernel on a grid represented by the matrix

$$\boldsymbol{a}_{5\times 5} = \begin{pmatrix} 0 & 0 & a_{-2,0} & 0 & 0 \\ 0 & a_{-1,-1} & a_{-1,0} & a_{-1,1} & 0 \\ a_{0,-2} & a_{0,-1} & a_{0,0} & a_{0,1} & a_{0,2} \\ 0 & a_{1,-1} & a_{1,0} & a_{1,1} & 0 \\ 0 & 0 & a_{2,0} & 0 & 0 \end{pmatrix}$$

since we supposed the $N_1 \times N_2$ grid was such that $N_1 > N_2$, by theorem F.2, we have that $\boldsymbol{\phi}_1$ is depending only in the first variable x_1 and is monotone in x_1. Recall from F.2 that

$$\boldsymbol{\phi}_1(i) = \cos\left(\frac{\pi i}{N_1} + \frac{\pi}{2N_1}\right)$$

The vector $\frac{N_1}{\pi}\nabla \arccos(\boldsymbol{\phi}_1)$ will be denoted by $\boldsymbol{F}_1$ in the rest. Notice all entries of $\boldsymbol{F}_1$ are 0 or ± 1. Denote by $\boldsymbol{F}_2$ the gradient vector $\frac{N_2}{\pi}\nabla \arccos(\boldsymbol{\phi}_k)$ where $\boldsymbol{\phi}_k$ is the eigenvector given by theorem F.2 that is depending only in the second variable x_2 and is monotone in x_1 and recall

$$\boldsymbol{\phi}_k(i) = \cos\left(\frac{\pi i}{N_2} + \frac{\pi}{2N_2}\right)$$

For a matrix $\boldsymbol{B}$, let $\boldsymbol{B}^{\pm}$ the positive/negative parts of $\boldsymbol{B}$, ie matrices with positive entries such that $\boldsymbol{B} = \boldsymbol{B}^{+} - \boldsymbol{B}^{-}$. Let $\boldsymbol{B}_{r1}$ be a matrix representing the radius 1 kernel with weights

$$\boldsymbol{a}_{3\times 3} = \begin{pmatrix} 0 & a_{-1,0} & 0 \\ a_{0,-1} & a_{0,0} & a_{0,1} \\ 0 & a_{1,0} & 0 \end{pmatrix}$$

The matrix $\boldsymbol{B}_{r1}$ can be obtained by theorem F.4. Then the radius 2 kernel k is defined by all the possible combinations of 2 positive/negative steps, plus the initial radius-1 kernel.

$$\boldsymbol{B}_{r2} = \sum_{\substack{-2\leq i,j\leq 2 \\ |i|+|j|=2}} \underbrace{\left(a_{i,j}(\boldsymbol{F}_1^{sgn(i)})^{|i|}(\boldsymbol{F}_2^{sgn(j)})^{|j|}\right)}_{\text{Any combination of 2 steps}} + \underbrace{\boldsymbol{B}_{r1}}_{\text{all possible single-steps}}$$

with sgn the sign function $sgn(i) = +$ if $i \geq 0$ and $-$ if $i < 0$. The matrix $\boldsymbol{B}_{r2}$ then realises the kernel $\boldsymbol{a}_{5\times 5}$.

We can further extend the above construction to N dimension grids and radius R kernels k

$$\underbrace{\sum_{\substack{V=\{v_1,v_2,...,v_N\}\in\mathbb{N}^n \\ ||V||_{L^1}\leq R \\ -R\leq v_i\leq R}}}_{\text{Any choice of walk } V \text{ with at most } R\text{-steps}} a_V \underbrace{\prod_{j=1}^{N} (\boldsymbol{F}_j^{sgn(v_j)})^{|v_j|}}_{\text{Aggregator following the steps defined in } V}$$

with $\boldsymbol{F}_j = \frac{N_j}{\pi}\nabla \arccos \boldsymbol{\phi}_j$,$\boldsymbol{\phi}_j$ the eigenvector with lowest eigenvalue only dependent on the j-th variable and given in theorem F.2 and $\prod$ is the matrix multiplication. V represents all the choices of walk $\{v_1, v_2, ..., v_n\}$ in the direction of the fields $\{\boldsymbol{F}_1, \boldsymbol{F}_2, ..., \boldsymbol{F}_n\}$. For example, $V = \{3, 1, 0, -2\}$ has a radius $R = 6$, with 3 steps *forward* of $\boldsymbol{F}_1$, 1 step *forward* of $\boldsymbol{F}_2$, and 2 steps *backward* of $\boldsymbol{F}_4$.

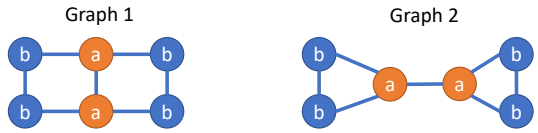
□

F.11 Proof for theorem 2.5 (Comparison with 1-WL test)

DGNs using the *mean* aggregator, any directional aggregator of the first Laplacian eigenvector and injective degree-scalers are strictly more powerful than the 1-WL test.

Proof. We will show that (1) DGNs are at least as powerful as the 1-WL test and (2) there is a pair of graphs which are not distinguishable by the 1-WL test which DGNs can discriminate.

Since the DGNs include the mean aggregator combined with at least an injective degree-scaler, Corso et al. (2020) show that the resulting architecture is at least as powerful as the 1-WL test.



Aggregation matrix	Graph 1	Graph 2
$\boldsymbol{A}$	$1a + 1b \rightarrow b$ $1a + 2b \rightarrow a$	$1a + 1b \rightarrow b$ $1a + 2b \rightarrow a$
$\boldsymbol{B}^1_{dx}$	$\lvert 1a - 1b \rvert \rightarrow b$ $0 \rightarrow a$	$\lvert 1a - 1b \rvert \rightarrow b$ $\lvert 0.44b - 0.44a \rvert \rightarrow a$
$\boldsymbol{B}^1_{av}$	$1a \rightarrow b$ $1b \rightarrow a$	$1a \rightarrow b$ $0.44b + 0.56a \rightarrow a$

Figure 8: Illustration of an example pair of graphs which the 1-WL test cannot distinguish but DGNs can. The table shows the node feature updates done at every layer. MPNN with mean/sum aggregators and the 1-WL test only use the updates in the first row and therefore cannot distinguish between the nodes in the two graphs. DGNs also use directional aggregators that, with the vector field given by the first eigenvector of the Laplacian matrix, provides different updates to the nodes in the two graphs.

Then, to show that the DGNs are strictly more powerful than the 1-WL test it suffices to provide an example of a pair of graphs that DGNs can differentiate and 1-WL cannot. Such a pair of graphs is illustrated in figure 8.

The 1-WL test (as any MPNN with, for example, sum aggregator) will always have the same features for all the nodes labelled with a and for all the nodes labelled with b and, therefore, will classify the graphs as isomorphic. DGNs, via the directional smoothing or directional derivative aggregators based on the first eigenvector of the Laplacian matrix, will update the features of the a nodes differently in the two graphs (figure 8 presents also the aggregation functions) and will, therefore, be capable of distinguishing them.

□

G APPENDIX - BROADER IMPACT

This work will extend the usability of graph networks to all problems with physically defined directions, thus making GNN a new laboratory for physics, material science and biology. In fact, the anisotropy present in a wide variety of systems could be expressed as vector field (spinor, tensor) compatible with the DGN framework, without the need of eigenvectors. One example is magnetic anisotropicity in metals, alloys and also in molecules such as benzene ring, alkene, carbonyl, alkyne that are easier or harder to magnetise depending on the directions or which way the object is rotated. Other examples are the response of material to high electromagnetic fields (e.g. to study material responses at terahertz frequency); all kind of field propagation in crystals lattices (vibrations, heat, shear and frictional force, young modulus, light refraction, birefringence); multi-body or liquid motion; traffic modeling; and design of novel materials and constrained structures. This also enables GNNs to be used for virtual prototyping systems since the added directional constraints could improve the analysis of a product's functionality, manufacturing and behavior.