# Hub-Laplacian operators for Directional GNNs

**Alessandro Carraro, Javier Gallardo Saenz, Ida Heijmink**
{a.carraro; j.gallardosaenz; i.m.heijmink}@student.tudelft.nl

## Abstract

Diffusion-based message passing is a widely used paradigm in graph neural networks (GNNs), but it faces challenges such as over-smoothing and over-squashing, which limit model expressivity. While recent approaches, including anisotropic diffusion and PDE-inspired advection-diffusion operators, aim to address these issues, the role of node degrees in shaping information flow remains underexplored. In this project, we investigate the hub Laplacian $L_\alpha$, a degree-biased graph shift operator. Depending on the parameter $\alpha$, $L_\alpha$ can attract ($\alpha > 0$) or repel ($\alpha < 0$) information from high-degree hubs. We integrate both fixed and learnable versions of $L_\alpha$ into existing GNN architectures, including models with anisotropic filters, and combine it with degree-biased advection operators. Our experiments show that, when $\alpha$ is properly tuned, the models employing the hub Laplacian consistently outperform standard baselines. These gains are likely due to improved spectral properties, such as larger spectral gaps.

## 1 Introduction

Diffusion-based message passing has become a popular paradigm for learning on graphs in recent years [11, 4, 6]. However, classical diffusion-based graph neural networks (GNNs) face two key limitations: over-smoothing [14, 16], where repeated aggregation causes node features to become indistinguishable, and over-squashing [2, 1], where long-range information is compressed through limited-capacity paths, leading to poor modeling of distant dependencies. Together, these issues restrict GNN expressivity.

To address these problems, recent works have proposed directing information propagation in graphs. Graph anisotropic diffusion (GAD) biases message flow along meaningful directions [5], PDE-inspired GNNs introduce discretized advection and advection–diffusion operators [13], and directional kernels based on Laplacian eigenvectors guide messages along vector fields [3].

In this project we explore alternative solutions combining the previous works with a new class of graph shift operators (GSO) based on advances in network sciences. Miranda et al. [12], Estrada et al. [7], and Gambuzza et al. [8] introduced the degree-biased Laplacian operator $\mathcal{L}_\alpha$ which in matrix form reads:

$$L_\alpha \;=\; \Xi_\alpha \;-\; D^\alpha A\, D^{-\alpha}, \quad \text{with } \Xi_\alpha = \operatorname{diag}\Big(\sum_{w\in\mathcal{N}(v)} (d_w/d_v)^\alpha\Big). \tag{1}$$

Here, $d_v$ is the degree of node $v$, $A$ is the adjacency matrix, and $D = \operatorname{diag}(d_v)$. Although asymmetric for $\alpha \neq 0$, $L_\alpha$ remains positive semidefinite and shifts the Laplacian spectrum [12]. A degree-biased advection operator can also be defined as the adjoint of the hub Laplacian. Combining it with the standard Laplacian leads to the advection-diffusion operator $T = \gamma_{adv} L^*_\alpha + \gamma_{diff} L$.

In this project, we incorporate the hub Laplacian into existing GNN architectures based on convolutional and anisotropic filters [5]. The goal is to introduce a degree-based bias that creates preferred flow paths, potentially enhancing information propagation. We evaluate whether this bias improves model performance compared to standard baselines.

Preprint. Under review.

## 2 Methodology

In our experiments we use two architectures; a graph convolutional neural network and a modified version of the GAD architecture proposed by Elhag et. al. [5], evaluated on the QM9[1] dataset. The implementation is available at `https://github.com/javier-gallardo-saenz/CS4350.git`.

### 2.1 Graph Convolutional Neural Network

The GCNN architecture, illustrated in Figure 1, stacks graph convolutional layers based on polynomial filters of a graph shift operator $S$, followed by non-linear activations:

$$\sigma\left(H(X)+B\right), \quad H(X)=\sum_{k=0}^{K} S^k X_{l-1} H_{lk} \tag{2}$$

where $B$ is a bias matrix and $H_{lk}$ are learnable weights.

A key component is the **global pooling layer**, which aggregates node features into a graph-level representation passed to a final MLP for prediction. We study three pooling strategies: **max**, **mean**, and **sum**.

We consider two model variants:

- **Fixed $\alpha$:** The degree-bias parameter $\alpha$ is treated as a hyperparameter and remains constant during training.
- **Learnable $\alpha$:** $\alpha$ is optimized during training, requiring the graph shift operator $S = L_\alpha$ to be recomputed at each epoch.

### 2.2 Graph Anisotropic Diffusion

Graph Anisotropic Diffusion (GAD) was introduced in [5]. In each layer, it performs one discrete step of $\frac{d}{dt}X_l = D^{-1}LX_l$ and then propagates information through message passing using the aggregators $\{mean, max, min, dx_1\}$ where $dx_1$ denotes the operation:

$$M_l = A\{[X_l]_j \mid j \in \mathcal{N}_i\} = [X_l, B_{dx_1}X_l]^t \quad \text{where } B_{dx_1} = \hat{F}_1 - \text{diag}(\sum_j \hat{F}_{:,j})$$

and $\hat{F}_1$ is the row normalized map induced by the gradient of the Laplacian's first eigenvector (fiedler vector) values along the graph, as defined in [3].

We expand this approach by allowing the use of $L_\alpha$ and $T$ in the diffusion step with or without learnable parameters, and the use of $F_{L_\alpha}$ or $F_T$ instead of $F$ - which are just the eigenmaps of the corresponding Hub Operator.

### 2.3 Experimental Setup

**GCNN** A preliminary grid search was conducted to select key architectural hyperparameters, including network depth and width. We explored various configurations and found that, for our problem, a relatively small network provided sufficient capacity. The final GCNN architecture uses two graph convolution layers with 64 hidden units each, two readout layers with 64 and 32 hidden units respectively, and ReLU activations throughout. We limit ourself to filters up to degree $k = 2$ given the limited sizes of the graphs in the dataset.

A second grid search examined the impact of the hub Laplacian configuration. Specifically, we varied the degree-bias parameter $\alpha \in \{-1, -0.5, 0, 0.5, 1.0\}$, which was either used as a fixed parameter or as an initial value when $\alpha$ was learnable. We also compared different graph shift operators, namely the Hub Laplacian $L_\alpha$ and the Advection Operator $T$, and evaluated three pooling strategies: max, sum, and mean pooling.

[1] `https://pytorch-geometric.readthedocs.io/en/2.6.1/generated/torch_geometric.datasets.QM9.html`

**GAD** The GAD model is much more computationally expensive. Due to local memory restrictions, we had to lower the batch size to 48 graphs per batch and get rid of the intra-layer tower structure used in the original paper. Thus, each layer has the same blocks as in [5, Figure 2]. To keep manageable training times, we expand from 11 node features to 50 using a linear layer and learn atomic embbedings into a 1d tensor of size 25 (in the original paper these parameters are set to 100 and 50, respectively). Due to time constraints, we focus on learning Property 0 of QM9. Every grid search is ran twice on two random subsets of the QM9 dataset to ensure more robust results.

A first grid search exlored the performance of the model as a function of the eigenmap $F$: $F_{L_\alpha}$ with $\alpha \in \{-1, -0.5, 0, 0.5, 1\}$, diffusion operator fixed to $L_0$. A second grid search explored the impact of the diffusion step choices for the optimal $F_{L_{\alpha^*}}$ determined in the first search: Diffusion Operator $\in \{L_\alpha, T_{\alpha,\gamma_{adv},\gamma_{diff}}\}$ with $(\alpha, \gamma_{adv}, \gamma_{diff}) \in \{(\alpha^*, 0.5, 0.5), \text{Learnable}\}$. Finally, the resulting best performing architecture was, on the one hand, evaluated against the same subset of the QM9 dataset used by our GCNN for comparison and, on the other hand, a small ablation study was performed on it.

**Training Details** Both the GCNN and the message passing network were trained using the Adam optimizer with an initial learning rate of $10^{-4}$ and a separate learning rate of $10^{-2}$ for $\alpha$. We used a batch size of 64, trained for up to 300 epochs, and employed early stopping with a patience of 30 epochs. Weight decay of $10^{-4}$ was applied for regularization. A learning rate scheduler[2] with a reduction factor of 0.5 and patience of 10 epochs was used. Batch Normalization [10] was applied after each layer in both models. The experiments were conducted on a dataset of 1200 graphs, split into 70% training, 15% validation, and 15% test sets. .

## 3 Results

**GCNN** As a *baseline* model we use the GCNN with hyperparameters listed in section 2 and the combinatorial Laplacian as GSO ($\alpha = 0$).

Table 1: Average Test MAE per Target and Pooling type for the baseline model.

| Target | Max | Sum | Mean |
|---|---|---|---|
| 0 | 1.0783 | 1.1981 | **0.9329** |
| 1 | 6.3131 | **3.7228** | 6.1030 |
| 2 | 0.6845 | 1.2688 | **0.5741** |

From these results, we observe that the choice of pooling layer has a significant impact on GCNN performance. In the baseline models, mean pooling achieved the best performance for targets 0 and 2, while sum pooling was optimal for target 1 (Table 1).

Table 2: Validation and Test MAE per target. Best results per target are in bold. The baseline corresponds to $\alpha = 0$

| Target | Validation MAE | | Test MAE | | Initial $\alpha$ | Pooling |
|---|---|---|---|---|---|---|
| | Best | Baseline | Best | Baseline | | |
| Target 0 | **0.9687** | 1.0025 | **0.7805** | 1.0407 | 0.5 (Learned) | Mean |
| Target 1 | **2.9704** | 3.2886 | **2.6446** | 3.3338 | 0.5 (Fixed) | Sum |
| Target 2 | **0.4952** | 0.5556 | **0.4269** | 0.5001 | 0 (Learned) | Mean |

Nevertheless, by selecting the optimal pooling strategy for each target and incorporating the hub Laplacian with either a fixed or learnable $\alpha$, we consistently outperformed the baseline. For instance, target 0 achieved its lowest MAE using mean pooling with $\alpha$ initialized at 0.5, which converged to approximately 0.98 during training, as shown in Figure 2. Notably, the top four models all consistently converged to $\alpha \approx 1$ for this target. Results involving the advection-diffusion operator were excluded

[2]https://docs.pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html

from the final grid search due to the difficulty in tuning the hyperparameters $\gamma_{\text{diff}}$ and $\gamma_{\text{adv}}$. These parameters require careful adjustment to remain small while properly balancing their contributions, as improper tuning can cause large eigenvalue magnitudes and thus poor convergence.

The experiments were also extended to the normalized Laplacian, defined as $\hat{L} = I - D^{-1/2}AD^{-1/2}$. Additionally, we introduced a normalized version of the hub Laplacian,

$$\hat{L}_\alpha = \Xi_\alpha - D^{-1/2-\alpha}AD^{-1/2+\alpha},$$

which reduces to the normalized Laplacian when $\alpha = 0$. Using these operators, we observed improved performance, with MAE even decreasing to 0.67 for target 0. This improvement is likely attributable to the enhanced spectral properties of the normalized operators [15]. However, due to time and computational constraints, these results are based on a limited set of runs and datasets, and thus are not statistically significant enough to draw definitive conclusions about the superiority of the normalized hub Laplacian.

**GAD** The results of the first grid search for the optimal GAD architecture confirmed our findings in 3: using $L_{0.5}$ produces the best results (3).

Table 3: Averaged MAE for Validation and Test Sets across two random subsets of QM9 for Different Alpha Values of $F_{L_\alpha}$

| **Metric** | $\alpha = -1$ | $\alpha = -0.5$ | $\alpha = 0$ | $\alpha = 0.5$ | $\alpha = 1$ |
|---|---|---|---|---|---|
| Run averaged Validation MAE | 0.7548 | 0.7494 | 0.7419 | **0.7379** | 0.7634 |
| Run averaged Test MAE | 0.8242 | 0.7869 | 0.8101 | **0.7387** | 0.8117 |

The subsequent grid search determined that the best diffusion step for $F_{L_{0.5}}$ is given by $T_{\alpha,\gamma_{adv},\gamma_{diff}}$ where the three parameters are learnt and their initial values set to $0.5$ (see 5).

The resulting architecture across the shuffled first $1200$ molecules of QM9 outperformed every other model. This is illustrated in 4.

Table 4: MAE for Validation and Test Sets across the shuffled first 1200 molecules of QM9 for our best performing models and the corresponding baselines

| **Metric** | GAD ($F_{0.5}$, learnable $T$) | GAD | GCNN (learnable $L_\alpha$) | GCNN |
|---|---|---|---|---|
| Validation MAE | **0.4427** | 0.4559 | 0.9687 | 1.0025 |
| Test MAE | **0.4389** | 0.4411 | 0.7805 | 1.0407 |

**Aggregators Ablation Study** To study the role of $F_{L_{0.5}}$ as an aggregator (more precisely, of the $B_{dx}$ obtained from it), we trained our best performing model with and without it on two random subsets (size 1200) of QM9. Surprisingly, the Average Test MAE only suffered a $0.0105$ increase when $B_{dx}$ was removed. This suggests that using $T$ in the diffusion step might introduce enough directionality in the information propagation. Such a result demands further study, as this insight could potentially help bypass the calculation of the eigenmaps - which could in turn improve the method's flexibility and scalability.

One limitation of our study is that the role of the **pooling** strategy is underexplored. Recommended future work therefore includes further investigation of pooling strategies, possibly exploring the potential of hierarchical pooling. Moreover, the architectures potential to limit over-smoothing could be quantified using the Dirichlet energy as defined in [14].

## 4 Discussion

The results of the GCNN experiments show that the degree-biased Laplacian with a positive $\alpha$ outperform the standard Laplacian on the QM9 dataset. Giraldo et. al. [9] studied the role of the spectral gap in over-smoothing and over-squashing on the Simple Graph convolution model (SGC), highlighting a trade-off. A small gap exacerbates squashing by trapping information, while a large gap accelerates smoothing. We observe that the hub Laplacian with a positive $\alpha = 0.5$ has a larger

spectral gap than the standard Laplacian 3, indicating that reducing over-squashing is more important than reducing over-smoothing in the problem at hand.

In analogy with signal processing, the graph convolution acts as spectral filter on the node features. A narrow spectral support (defined as the difference between the largest and the smallest eigenvalue) favors low-frequency signals while a larger support enables retention of higher-frequency components, preventing rapid smoothing. Comparing the spectral support of the two operators $\mathcal{L}_{0.5}$ and $\mathcal{L}_0$ reveals that the spectral support is in general smaller for $\mathcal{L}_{0.5}$, which implies stronger smoothing tendency. The fact that we do not observe over-smoothing could be explained by the relativly shallow model that we use (two convolutional layers).

The effect of the introduction of the Hub Operators in the GAD scheme is more challenging to understand, as this architecture is far less interpretable than a GCNN. Nevertheless, it is remarkable how the use of the graph shift operator made the model significantly more adaptable/robust to training over small, random subsets of QM9 - as revealed in 3. That the modified GAD architecture is able to predict different results from the GAD baseline is not surprising when one looks at the cosine similarity between the $F$ matrices generated by $L$ and $L_{0.5}$, which is almost 0 on average across the dataset.

### 4.1 Scalability

**GCNN** The total per-epoch computational complexity of the GCNN is given by $\mathcal{O}\left(Lk \times (NF^2 + \|S\|_0 F) + N + E\right)$, where $L$ is the number of layers, $k$ is the filter order, $N$ is the number of nodes, $F$ is an upper bound on the number of features, and $\|S\|_0$ is the number of non-zero entries in the graph shift operator (GSO). The first term accounts for the cost of graph filtering and feature transformations across all layers and filter orders, while the $O(N + E)$ term represents the cost of reconstructing the GSO at each epoch when $\alpha$ is a learnable parameter.

**GAD** Precomputing 1 eigenvector for a sparse matrix is $\mathcal{O}(|E|)$, so the precomputation of $F$ is just $\mathcal{O}(|V| + |E|)$. For each layer, let $B$ the number of nodes in the batch. Expanding the input features and atomic number from size $F$ to size $H$ is $\mathcal{O}(BFH)$. Computing one timestep of the diffusion equation using a standard sparse solver is around $\mathcal{O}(HB^{1.5})$ (Sparse Cholesky is more efficient but cannot be used if the diffusion operator is $L_\alpha$ or $T$). Finally, the complexity of the message passing module is dictated by the $dx_1$ aggregator which is $\mathcal{O}(|E_B|H)$. Learnable diffusion requires the in-situ generation of the gso, which adds an additional $\mathcal{O}(B + |E_B|)$ to the layer (here $|E|_B$ is the number of nonzero entries in the batch adjacency matrix).

## 5 CRediT author statement

**A.C** conceptualization; methodology; software; investigation; formal analysis; writing. **J.G** conceptualization, methodology; software; investigation; writing. **I.H** conceptualization, methodology; software; investigation; formal analysis; writing.

## References


[1] Singh Akansha. Over-squashing in graph neural networks: A comprehensive survey. URL http://arxiv.org/abs/2308.15568.

[2] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. URL http://arxiv.org/abs/2006.05205.

[3] Dominique Beaini, Saro Passaro, Vincent Létourneau, William L. Hamilton, Gabriele Corso, and Pietro Liò. Directional graph networks. URL http://arxiv.org/abs/2010.02863.

[4] Benjamin Paul Chamberlain, James Rowbottom, Maria I. Gorinova, Stefan Webb, Emanuele Rossi, and Michael M. Bronstein. GRAND: graph neural diffusion. *CoRR*, abs/2106.10934, 2021. URL https://arxiv.org/abs/2106.10934.

[5] Ahmed A. A. Elhag, Gabriele Corso, Hannes Stärk, and Michael M. Bronstein. Graph anisotropic diffusion, 2022. URL https://arxiv.org/abs/2205.00354.

[6] Moshe Eliasof, Eldad Haber, and Eran Treister. Graph neural reaction diffusion models. *SIAM Journal on Scientific Computing*, 46(4):C399–C420, 2024. doi: 10.1137/23M1576700. URL https://doi.org/10.1137/23M1576700.

[7] Ernesto Estrada and Delio Mugnolo. Hubs-biased resistance distances on graphs and networks. 507 (1):125728. ISSN 0022247X. doi: 10.1016/j.jmaa.2021.125728. URL http://arxiv.org/abs/2101.07103.

[8] Lucia Valentina Gambuzza, Mattia Frasca, and Ernesto Estrada. Hubs-attracting laplacian and related synchronization on networks. 19(2):1057–1079. ISSN 1536-0040. doi: 10.1137/19M1287663. URL https://epubs.siam.org/doi/10.1137/19M1287663.

[9] Jhony H. Giraldo, Konstantinos Skianis, Thierry Bouwmans, and Fragkiskos D. Malliaros. On the Trade-off between Over-smoothing and Over-squashing in Deep Graph Neural Networks. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pages 566–576, October 2023. doi: 10.1145/3583780.3614997. URL http://arxiv.org/abs/2212.02374. arXiv:2212.02374 [cs].

[10] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015. URL https://arxiv.org/abs/1502.03167.

[11] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. *CoRR*, abs/1911.05485, 2019. URL http://arxiv.org/abs/1911.05485.

[12] Manuel Miranda and Ernesto Estrada. Degree-biased advection–diffusion on undirected graphs/networks. 17:30. ISSN 0973-5348, 1760-6101. doi: 10.1051/mmnp/2022034. URL https://www.mmnp-journal.org/10.1051/mmnp/2022034.

[13] Yifan Qu, Oliver Krzysik, Hans De Sterck, and Omer Ege Kara. First-order pdes for graph neural networks: Advection and burgers equation models, 2024. URL https://arxiv.org/abs/2404.03081.

[14] T. Konstantin Rusch, Michael M. Bronstein, and Siddhartha Mishra. A survey on oversmoothing in graph neural networks. URL http://arxiv.org/abs/2303.10993.

[15] Mingqi Yang, Yanming Shen, Rui Li, Heng Qi, Qiang Zhang, and Baocai Yin. A new perspective on the effects of spectrum in graph neural networks, 2022. URL https://arxiv.org/abs/2112.07160.

[16] Weichen Zhao, Chenguang Wang, Xinyan Wang, Congying Han, Tiande Guo, and Tianshu Yu. Understanding oversmoothing in diffusion-based gnns from the perspective of operator semigroup theory, 2025. URL https://arxiv.org/abs/2402.15326.

# A Appendix

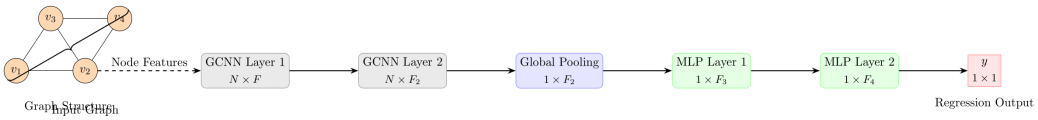


Figure 1: Schematic GCNN architecture

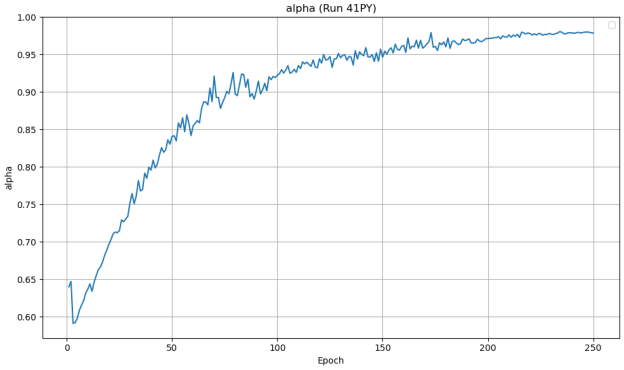


Figure 2: Evolution of $\alpha$ during training of best model for target 0 presented in Table 2

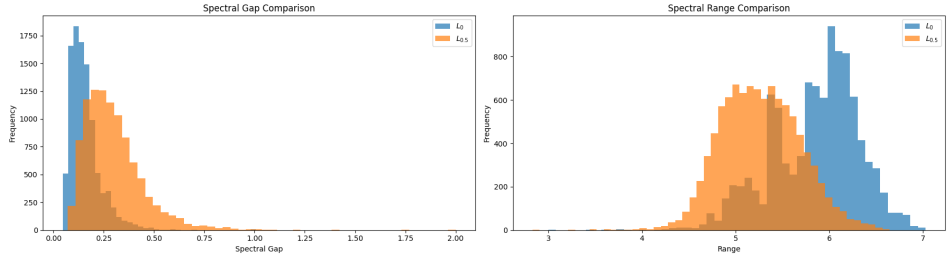


Figure 3: Spectral gap and spectral support distribution of the laplacian and hub laplacian ($\alpha = 0.5$) for the first 10k graphs in the dataset

Table 5: Mean Absolute Errors (MAE) for Different Diffusion Methods given $F_{L_{0.5}}$ and initialising $\alpha = \gamma_{adv} = \gamma_{diff} = 0.5$

| **Metric** | **Learnable $L_\alpha$** | **Learnable $T$** | **Non-learnable $L_\alpha$** | **Non-learnable $T$** | **Non-learnable $L$** |
|---|---|---|---|---|---|
| Val MAE | 0.7539 | **0.7434** | 0.7623 | 0.7507 | 0.7520 |
| Test MAE | 0.6347 | **0.6214** | 0.6655 | 0.6277 | 0.6599 |