GRAPH ANISOTROPIC DIFFUSION

Anonymous authors
Paper under double-blind review

ABSTRACT

The capacity of existing graph neural networks (GNNs) is limited by the use of local or isotropic kernels. We present Graph Anisotropic Diffusion: a new GNN architecture that utilizes a diffusion PDE on graphs. The main ingredient in our model is a linear diffusion layer with two solving schemes that is applied independently to each feature channel of the vertices in the graph. This learned diffusion layer improves the message passing mechanism in GNNs by allowing continuous propagation of information among nodes with control over the diffusion step. This diffusion layer is combined with local anisotropic kernels to obtain a notion of direction. Empirically we demonstrate the capacity of our model to improve the performance of competitive GNNs in two common molecular property prediction benchmarks.

1 INTRODUCTION

Graph neural networks (GNNs) have shown success in solving many real-world problems in different domains, including social networks & recommendation systems (Ying et al., 2018; Fan et al., 2019), chemistry (Gilmer et al., 2017; Sanchez-Lengeling et al., 2019), bioinformatics (Lv et al., 2021; Gainza et al., 2020), and drug discovery (Gaudelet et al., 2021). Graph inputs present several challenges because they require algorithms to reason about the underlying structure of the graph as well as its node and edge features.

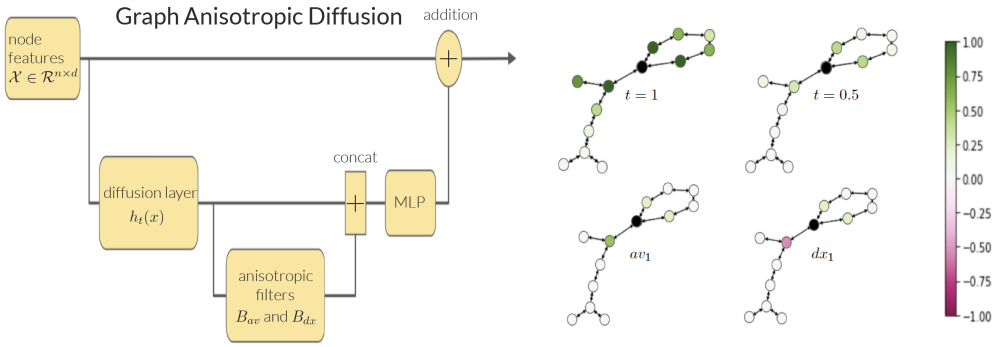
Most GNNs are based on the message passing mechanism (Gilmer et al., 2017), in which the features of each node in the graph are updated by aggregating information from its neighbours. However, recent studies have demonstrated several limitations of this message passing paradigm, which include limited expressive power (Xu et al., 2019; Corso et al., 2020; Bodnar et al., 2021), as well as the oversmoothing (Oono & Suzuki, 2021) and bottleneck phenomena (Alon & Yahav, 2021).

To resolve some of these limitations: recent work has integrated diffusion processes in GNNs. For example, Graph Diffusion Convolution (GDC) (Klicpera et al., 2019) showed empirical improvements over the traditional message passing mechanism. GRAND (Chamberlain et al., 2021b) and BLEND (Chamberlain et al., 2021a) leverage diffusion PDEs to propagate messages continuously on the graph. However, they rely on solving nonlinear systems of PDEs numerically, which leads to empirical limitations. Separate works (Beaini et al., 2021), instead, have linked the mentioned problems of message passing to the structural isotropicity of the messages passed proposing a simple way of using the spectral decomposition of the graph to impose directionality to local messages.

In contrast, combining anisotropicity with diffusion has shown competitive results for learning on meshes and point clouds, achieving state-of-the-art on multiple 3D vision tasks (Sharp et al., 2021). In this work, we show that similar ideas can be applied to graphs. We propose a new efficient way of doing isotropic diffusion on graphs with learnable kernel size, combined with a local anisotropic diffusion operator. Thus, we can obtain a global anisotropic diffusion over the graph.

The main ingredient in our model is a linear diffusion layer with a learnable parameter t that is applied independently to each feature channel in the graph. We present two simple and efficient solving schemes for our linear diffusion layer; implicit timestep and spectral expansion. We combine this diffusion with the anisotropic filters proposed by Beaini et al. (2021) in a novel architecture that we call Graph Anisotropic Diffusion (GAD).

Empirically, we show that GAD improves the performance of competitive GNNs on two popular molecular property prediction benchmarks: ZINC and QM9.



(a) Graph Anisotropic Diffusion Block

(b) Diffusion kernels and isotropic filters

Figure 1: Fig(a): One block of Graph Anisotropic Diffusion. Fig (b): The top two graphs show the diffusion kernels for different t values for a particular node (the black node), while the bottom two graphs show the local directional derivative and directional smoothing for this node.

2 Background and Related work

Diffusion equation. Diffusion describes the movement of a quantity from regions of high concentration to lower concentration over time. A natural example is the spread of temperature: when we put a hot object in a cold environment, the heat will be transferred from the object to the environment up to the point of equilibrium. A PDE that describes this process is called the heat equation, which is defined in equation 1. In this equation, $x_t \in [0, \infty]$, is a family of scalar-valued functions defined on the domain Ω, that represents the distribution of temperature at some time, and $x_t(u)$ will be its value at point $u \in \Omega$ at time t. Δx_t is the temperature gradient; or the Laplacian of the function $x_t(.) : \Omega \to R$. Starting from initial distribution x_0, the solution of equation 1 is $x_t = \exp(-t\Delta)x_0$, as t goes from 0 to ∞ until reaching some equilibrium point over the domain Ω.

$$\frac{d}{dt}x_t = \Delta x_t \tag{1}$$

Image processing and computer vision. A lot of work in the literature on image processing is related to diffusion equations. For example, the geometrical framework for image enhancement proposed by Sochen et al. (1998). Noniterative Bilateral filtering by Tomasi & Manduchi (1998) and it is related to anisotropic diffusion by Durand & Dorsey (2002). In 3D vision tasks, Sharp et al. (2021) discretised the diffusion equation defined in 1 by replacing Δ with the weak Laplacian matrix $\mathcal{L}$. Then $\mathcal{L}$ is accompanied by a mass matrix $\mathcal{M}$, such that the diffusion rate is given by $\mathcal{M}^{-1}\mathcal{L}x$. In addition, Sharp et al. (2021) obtained directional anisotropic filters by combining the above diffusion with local spatial gradient features obtained from the inner product between pairs of feature gradients at each node on meshes. This combination of global isotropic diffusion and local anisotropic information achieves a very effective family of global anisotropic filters for learning on meshes and point clouds. We will use a similar approach to obtain such filters on graphs.

Neural differential equations on graphs. In multiple works, such as deep residual learning on graphs through ODEs by Avelar et al. (2019), neural ODEs for static and dynamic graphs by Poli et al. (2021), continuous GNNs by Xhonneux et al. (2020); Gu et al. (2021), graph neural networks have been modelled as discretisations of the diffusion equation. Another similar work also based on PDEs is GRAND, proposed by Chamberlain et al. (2021b), in which they introduced graph neural networks as discretisations of the diffusion equation. Non-Euclidean extension to GRAN is BLEND, which was also developed by Chamberlain et al. (2021a).

Anisotropic kernels on graphs. Our method builds upon DGN proposed by Beaini et al. (2021) to give some notion of directions on graphs. We develop a linear discretisation of the diffusion equation using two simple and efficient solving schemes. Differ from previous methods such as GRAND and BLEND (Chamberlain et al., 2021b;a), in which they solved the nonlinear PDE numerically.

3 GRAPH ANISOTROPIC DIFFUSION

In this section, we present our Graph Anisotropic Diffusion (GAD) method. Firstly, we show how an efficient graph diffusion kernel can be obtained by discretising the diffusion equation with a linear closed-form expression. We then describe the local anisotropic kernels used to provide some notion of direction. Finally, we present how these two components fit together in GAD. Figure 1a provides a high-level overview of the GAD architecture layer. And Figure 1b gives some visualisation for different diffusion kernels and isotropic filters.

3.1 LEARNED DIFFUSION

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph with $|\mathcal{V}| = n$ nodes and $|\mathcal{E}| = e$ edges, and let $\mathcal{A} \in \mathcal{R}^{n\times n}$ be the adjacency matrix of $\mathcal{G}$, and $\mathcal{D} \in \mathcal{R}^{n\times n}$ is the node degree matrix; a diagonal matrix whose entities are the degrees of nodes in the graph $\mathcal{G}$. The Laplacian matrix $\mathcal{L} \in \mathcal{R}^{n\times n}$ of the graph $\mathcal{G}$ is a sparse negative semi-definite matrix defined by $\mathcal{L} = \mathcal{D} - \mathcal{A}$.

Our method is based on the discretisation of the diffusion equation defined in 1, using a linear expression to propagate features for learning on graphs instead of solving it numerically. We modify equation 1 by replacing Δ with the weak Laplacian matrix $\mathcal{L}$. Then accompany it with the node degree matrix $\mathcal{D}$, such that the diffusion rate is given by $\mathcal{D}^{-1}\mathcal{L}x$, as mentioned in section 2.

Let us assume the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with n nodes has a d-dimensional node-wise features matrix, $\mathcal{X} \in \mathcal{R}^{n\times d}$. We propose a learned diffusion layer $h_t : R^d \to R^d$ which applied independently to each feature channel with a separate learned time t per channel. This enables us to go beyond traditional message passing mechanisms in GNNs. Instead of a single propagation step or discrete random walk, we perform a continuous diffusion with control over the parameter t that determines the kernel size and can be learned end-to-end via SGD. As a result, we can continuously optimise the propagation of the features at each node from purely local to completely global, depending on the diffusion time t.

How to compute the diffusion? Different numerical schemes are available to solve the proposed linear diffusion operation. To learn and optimise the parameter t via back-propagation, we should confine ourselves to differentiable schemes. In this regard, two simple schemes are used in the experiments: *implicit timestep* and *spectral expansion*. The implementation of both is simple using common scientific libraries.

Direct Implicit Timestep. A simple way to evaluate the diffusion is to use the implicit Euler timestep, which is given by equation 2. Implicit timestep requires solving sparse linear system equations for each diffusion operation, which we solved using Cholesky decomposition.

$$h_t(x) = (\mathcal{D} + t\mathcal{L})^{-1}\mathcal{D}x \tag{2}$$

Spectral Scheme. This method approximates the diffusion equation by a closed-form expression using the low-frequency Laplacian eigenfunctions. The diffusion is calculated using the k lowest eigenvalues and the corresponding eigenvectors of the Laplacian matrix. Let us define the eigenvectors $\phi_i \in \mathcal{R}^n$ by $\mathcal{L}\phi_i = \lambda_i\mathcal{D}\phi_i$, where $\mathcal{L}$ is the weak Laplacian matrix, and $\mathcal{D}$ is the node degree matrix. Consequently, equation 3 approximates the diffusion equation, in which $\odot$ denotes the Hadamard product (elementwise), and ϕ denotes the stacked matrix of the k eigenvectors of $\mathcal{L}$.

$$h_t(x) = \phi \begin{bmatrix} e^{-\lambda_0 t} \\ e^{-\lambda_1 t} \\ \dots \end{bmatrix} \odot \phi^T\mathcal{D}x \tag{3}$$

Calculating the k lowest eigenvalues and the corresponding eigenvectors can be a preprocessing step as it does not depend on the features x or parameter t. Therefore, with a small approximation error, we obtain a significantly more efficient method.

3.2 ANISOTROPIC FILTERS

The anisotropic filters on graphs proposed by Beaini et al. (2021), in which they used a vector field to define multiple aggregators, which gives directions for the propagation of information on the graphs. We used directional smoothing and directional derivative aggregators to define the anisotropic kernels as the second ingredient in our architecture.

The vector field F is defined on graphs by $F = \Delta\phi$, where ϕ is the eigenvectors of the Laplacian matrix $\mathcal{L}$ that have the lowest eigenvalues, and $\Delta\phi_{ij} = \phi_i - \phi_j$. By normalising each row in F by L^1 norm and taking the absolute value, we can drive the directional smoothing aggregation matrix B_{av} as in equation 4 (ϵ is just to avoid the floating-point errors). Additionally, by subtracting a diagonal matrix whose entities contain the summation of each column in F from the normalised vector field $\hat{F}$, we will get the directional derivative aggregation matrix B_{dx} given in equation 5.

$$B_{av}(F_{i,:}) = \frac{|F_{i,:}|}{||F_{i,:}||_{L^1} + \epsilon} \quad (4) \qquad B_{dx}(F_{i,:}) = \hat{F}_{i,:} - diag(\sum_j \hat{F}_{:,j})_{i,:} \quad (5)$$

4 Experiments and results

To evaluate the expressiveness of our model, we tested on two popular benchmark datasets: ZINC Irwin et al. (2012) and QM9 Ramakrishnan et al. (2014). Our results on each dataset are presented with a comparison with previous models in Tables 1 and 2. For the ZINC dataset, we used the same number of parameters (around 100k) used in previous baselines. For QM9, we tested our model on the first seven properties provided by the PyTorch Geometric Fey & Lenssen (2019) library keeping again constant the number of parameters and removing the 3D coordinates from the model's inputs, as often done in the literature. More implementation details and hyperparameter settings for ZINC and QM9 are provided in appendices A and B.

Given that DGN is the baseline GNN that our model was constructed upon, we consider this as the main comparison point to analyse the effectiveness of the diffusion process presented. On ZINC, the addition of the diffusion layer reduces by more than 17% the mean absolute error, and on QM9, it improves the performance in every single property. Note that while methods using the addition of tailored features or message passing structure achieve better performances on ZINC, these improvements are orthogonal to the ones proposed in this work and can therefore be combined.

Table 1: Mean absolute error (MAE): ZINC. GAD-i uses the implicit timestep. GAD-s uses the spectral expansion scheme. *uses additional tailored features such as cycles.

Model	No edge features	Edge features
GCN	0.469± 0.002	-
GIN	0.408± 0.008	-
GraphSage	0.410± 0.005	-
GAT	0.463± 0.002	-
MoNet	0.407± 0.007	-
GatedGCN	0.422± 0.006	0.363 ± 0.009
PNA	0.320± 0.032	0.188± 0.004
DGN	0.219± 0.010	0.168± 0.003
HIMP*	-	0.151± 0.006
SMP*	0.219±	0.138±
GSN*	0.140± 0.006	0.115± 0.012
GAD-i	0.181± 0.004	0.139± 0.007
GAD-s	0.194± 0.006	0.140± 0.004

Table 2: Mean absolute error (MAE): QM9 properties.

	Unit	PNA	DGN	**GAD-s**
μ	D	0.365	0.354	0.338
α	a_0^3	0.255	0.250	0.240
ϵ_{HOMO}	meV	71.5	71.1	64.3
ϵ_{LUMO}	meV	70.1	68	63
$\Delta\epsilon$	meV	100.2	99.2	96.8
$\langle R^2 \rangle$	${a_0}^2$	18.3	17.2	16.5
ZPVE	meV	6.57	6.03	4.84

5 Conclusion

We proposed Graph Anisotropic Diffusion (GAD), a novel method that consists of a new efficient way of performing isotropic diffusion on graphs combined with a local anisotropic kernel. This approach is used to obtain a global anisotropic diffusion. Empirical results on popular graph benchmarks confirm the improved capacity of our method. We hope this work opens up the opportunity for future research directions in alternative methods to obtain anisotropic diffusion on graphs and theoretical analysis of their expressive power.

REFERENCES

Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications, 2021.

Pedro H. C. Avelar, Anderson R. Tavares, Marco Gori, and Luis C. Lamb. Discrete and continuous deep residual learning over graphs, 2019.

Muhammet Balcilar, Pierre Héroux, Benoit Gaüzère, Pascal Vasseur, Sébastien Adam, and Paul Honeine. Breaking the limits of message passing graph neural networks, 2021.

Dominique Beaini, Saro Passaro, Vincent Létourneau, William L. Hamilton, Gabriele Corso, and Pietro Liò. Directional graph networks, 2021.

Cristian Bodnar, Fabrizio Frasca, Yu Guang Wang, Nina Otter, Guido Montúfar, Pietro Liò, and Michael Bronstein. Weisfeiler and lehman go topological: Message passing simplicial networks, 2021.

Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M. Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting, 2021.

Xavier Bresson and Thomas Laurent. Residual gated graph convnets, 2018.

Marc Brockschmidt. GNN-FiLM: Graph neural networks with feature-wise linear modulation. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 1144–1152. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/brockschmidt20a.html.

Benjamin Paul Chamberlain, James Rowbottom, Davide Eynard, Francesco Di Giovanni, Xiaowen Dong, and Michael M Bronstein. Beltrami flow and neural diffusion on graphs, 2021a.

Benjamin Paul Chamberlain, James Rowbottom, Maria Gorinova, Stefan Webb, Emanuele Rossi, and Michael M. Bronstein. Grand: Graph neural diffusion, 2021b.

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

Frédo Durand and Julie Dorsey. Fast bilateral filtering for the display of high-dynamic-range images. *ACM Trans. Graph.*, 21(3):257–266, jul 2002. ISSN 0730-0301. doi: 10.1145/566654.566574. URL https://doi.org/10.1145/566654.566574.

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

Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation, 2019.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric, 2019.

Matthias Fey, Jan-Gin Yuen, and Frank Weichert. Hierarchical inter-message passing for learning on molecular graphs, 2020.

P. Gainza, F. Sverrisson, F. Monti, E. Rodolà, D. Boscaini, M. M. Bronstein, and B. E. Correia. Deciphering interaction fingerprints from protein molecular surfaces using geometric deep learning. *Nat Methods 17*, 52(7):184–192, 2020.

Thomas Gaudelet, Ben Day, Arian R. Jamasb, Jyothish Soman, Cristian Regep, Gertrude Liu, Jeremy B. R. Hayter, Richard Vickers, Charles Roberts, Jian Tang, David Roblin, Tom L. Blundell, Michael M. Bronstein, and Jake P. Taylor-King. Utilising graph machine learning within drug discovery and development, 2021.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry, 2017.

Fangda Gu, Heng Chang, Wenwu Zhu, Somayeh Sojoudi, and Laurent El Ghaoui. Implicit graph neural networks, 2021.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2018.

John J. Irwin, Teague Sterling, Michael M. Mysinger, Erin S. Bolstad, and Ryan G. Coleman. Zinc: A free tool to discover chemistry for biology. *Journal of Chemical Information and Modeling*, 52 (7):1757–1768, 2012.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning, 2019.

Guofeng Lv, Zhiqiang Hu, Yanguang Bi, and Shaoting Zhang. Learning unknown from correlations: Graph neural network for inter-novel-protein interaction prediction, 2021.

Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns, 2016.

Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification, 2021.

Michael Poli, Stefano Massaroli, Junyoung Park, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Graph neural ordinary differential equations, 2021.

Raghunathan Ramakrishnan, Pavlo O. Dral, Matthias Rupp, and O. Anatole von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data*, 1(1):1–7, 2014.

Benjamin Sanchez-Lengeling, Jennifer N. Wei, Brian K. Lee, Richard C. Gerkin, Alán Aspuru-Guzik, and Alexander B. Wiltschko. Machine learning for scent: Learning generalizable perceptual representations of small molecules, 2019.

Nicholas Sharp, Souhaib Attaiki, Keenan Crane, and Maks Ovsjanikov. Diffusionnet: Discretization agnostic learning on surfaces, 2021.

N. Sochen, R. Kimmel, and R. Malladi. A general framework for low level vision. *IEEE Transactions on Image Processing*, 7(3):310–318, 1998. doi: 10.1109/83.661181.

C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, pp. 839–846, 1998. doi: 10.1109/ICCV.1998.710815.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

Clement Vignac, Andreas Loukas, and Pascal Frossard. Building powerful and equivariant graph neural networks with structural message-passing, 2020.

Louis-Pascal A. C. Xhonneux, Meng Qu, and Jian Tang. Continuous graph neural networks, 2020.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks?, 2019.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, Jul 2018. doi: 10.1145/3219819.3219890. URL http://dx.doi.org/10.1145/3219819.3219890.

A IMPLEMENTATION DETAILS

The general framework of a graph neural network architecture on which we place our components, defined by Gilmer et al. (2017) ; in equations 6 and 7. Here k is the layer index, h_i^{k+1} is the features vector of node i at layer k, h_j^k is the features vector of node j which is the neighborhood of i, e_{ij} is the edge features vector between nodes i and j (optional), M_k is the massage passing function of layer k, U_k is a multi-layer perceptron (MLP), and $\bigoplus$ is just an operator that concatenates the output of multiple aggregators M_k. We can see clearly our architecture in this framework, hence B_{dx} and B_{av} are massage passing functions (M_k), and the diffusion layer is a linear transformation.

$$h_i^{k+1} = U_k(h_i^k, m_i^{k+1}) \tag{6}$$

$$m_i^{k+1} = \bigoplus_{(i,j)\in E} M_k(h_i^k, h_j^k, e_{ij}) \tag{7}$$

We used also mean, max, and min aggregators, in addition to degree scalars $S(d, \delta)$ proposed by PNA (Corso et al., 2020) which is defined in equations 8 and 9, where d is the degree of the node, δ is the average node degree in the training data, and α is the scaling parameter ($-1 \leq \delta \leq 1$).

$$\delta = \frac{1}{|train|} \sum_{i \in train} \log(d_i + 1) \tag{8}$$

$$S(d, \alpha) = \frac{\log(d+1)}{\delta} \tag{9}$$

B DATASETS AND HYPERPARAMETER SETTING

B.1 ZINC

ZINC is a molecule graphs dataset in which the task is to predict the penalized water-octanol partition coefficient of a certain molecule (graph regression). We used the same split provided by Dwivedi et al. (2020) ; 10k training, 1k validation, and 1k test, with mean absolute error (MAE) evaluation metric. We compare our model against various GNNs architectures, namely GCN (Kipf & Welling, 2017), GIN (Xu et al., 2019), GraphSage (Hamilton et al., 2018), GAT (Veličković et al., 2018), MoNet (Monti et al., 2016), GatedGCN (Bresson & Laurent, 2018), PNA (Corso et al., 2020), DGN (Beaini et al., 2021), GNNML (Balcilar et al., 2021), HIMP (Fey et al., 2020), SMP (Vignac et al., 2020), and GSN (Bouritsas et al., 2021). We did five runs using each solving scheme (implicit timestep 3.1 and spectral expansion 3.1).

The total number of parameters used in GAD for the results in Table 1 is the same as the one used in DGN ($\approx$ 100K). We also used the same hyperparameters setting in DGN, just decreasing the hidden dimensions to keep the same number of parameters.

B.2 QM9

QM9 is a common molecule graph benchmark, which provides different quantum mechanical properties of small molecules. It has various node and edge features. We used all of them except the atomic coordinates of the nodes. For the data split, we used around 110k training, 10k validation, and 10k test samples as provided by Brockschmidt (2020). The evaluation metric is mean absolute error (MAE). Due to computational constraints, we did one run on the first seven properties provided by the PyTorch Geometric Fey & Lenssen (2019) library using the spectral expansion scheme for the results in Table 2. For comparison, we did also one run using PNA (Corso et al., 2020) and DGN (Beaini et al., 2021) models.

For QM9, we fine tuned GAD over the following hyperparameters setting:

- Weight decay $\in \{3 \times 10^{-3}, 3 \times 10^{-4}, 3 \times 10^{-5}, 3 \times 10^{-6}, 3 \times 10^{-7}\}$.

- Dropout $\in \{0, 0.1, 0.2, 0.3\}$.
- $lr \in \{1 \times 10^{-3}\}$.
- Batch size $\in \{128, 256\}$.
- Number of layers $\in \{6, 7, 8, 9, 10\}$.
- Hidden dimensions $\in \{75, 90, 130, 150\}$.
- Aggregators $\in \{(mean, av_1, dx_1), (mean, max, min, dx_1), (mean, max, min, av_1, dx_1), (mean, sum, max, dx_1)\}$.

For PNA (Corso et al., 2020) and DGN (Beaini et al., 2021), we ran each one with the hyperparameters obtained from the search above, and increased the number of hidden dimensions to have the same number of parameters. Aggregators used in PNA are $(mean, max, min, std)$, and in DGN are $(mean, sum, max, dx_1)$.