Directional Graph Networks – Appendix [1]

A. Appendix - Choices of directional aggregators

This appendix helps understand the choice of $\boldsymbol{B}_{av}$ and $\boldsymbol{B}_{dx}$ in section 2.4 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.

A.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 $\boldsymbol{D}$

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

A.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 matrix-norm instead of a vector-norm. Doing so will modulate the aggregation by the amplitude of the field at each node. One needs to be careful since a global normalization might be very sensitive to the number of nodes in the graph.

A.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.

$$\boldsymbol{B}_{av-center}(\boldsymbol{F})_{i,:} = \frac{\boldsymbol{F}'^{+}_{i,:} + \boldsymbol{F}'^{-}_{i,:}}{||\boldsymbol{F}'^{+}_{i,j} + \boldsymbol{F}'^{-}_{i,j}||_{L_1}} \quad , \qquad \boldsymbol{F}'^{\pm}_{i,:} = \frac{|\boldsymbol{F}^{\pm}_{i,:}|}{||\boldsymbol{F}^{\pm}_{i,:}||_{L^1} + \epsilon} \tag{11}$$

The same idea can be applied to the derivative aggregator equation 12 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.

[1] Due to the large size of some equations in the appendix, we keep the appendix in a single column format.

$\boldsymbol{F}^+$ 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.

$$\boldsymbol{B}_{dx\text{-center}}(\boldsymbol{F})_{i,:} = \boldsymbol{F}'_{i,:} - \text{diag}\left(\sum_j \boldsymbol{F}'_{:,j}\right)_{i,:} \quad , \quad \boldsymbol{F}'_{i,:} = \frac{1}{2}\left(\underbrace{\frac{\boldsymbol{F}^+_{i,:}}{||\boldsymbol{F}^+_{i,:}||_{L^1} + \epsilon}}_{\text{forward field}} + \underbrace{\frac{\boldsymbol{F}^-_{i,:}}{||\boldsymbol{F}^-_{i,:}||_{L^1} + \epsilon}}_{\text{backward field}}\right) \tag{12}$$

A.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 $\boldsymbol{F}_{\text{softhard}}$.

$$(\boldsymbol{F}_{\text{softhard}})_{i,:} = \text{sign}(\boldsymbol{F}_{i,:})\text{softmax}(T|\boldsymbol{F}_{i,:}|) \tag{13}$$

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.

$$(\boldsymbol{F}_{\text{hard}})_{i,:} = \text{sign}(\boldsymbol{F}_{i,:})\text{argmax}(|\boldsymbol{F}_{i,:}|) \tag{14}$$

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

A.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.

$$\boldsymbol{F}_{\text{forward}} = \boldsymbol{F}^+ \quad , \qquad \boldsymbol{F}_{\text{backward}} = \boldsymbol{F}^- \tag{15}$$

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.

$$\boldsymbol{F}_{\text{forward copy}} = \boldsymbol{F}^+_{\text{hard}} \quad , \qquad \boldsymbol{F}_{\text{backward copy}} = \boldsymbol{F}^-_{\text{hard}} \tag{16}$$

A.6. Phantom zero-padding

Some recent work in computer vision has 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.

$$(\boldsymbol{B}_{av,0pad})_{i,:} = \frac{1}{2}\left(\frac{|\boldsymbol{F}^+_{i,:}|}{||\boldsymbol{F}^+_{i,:}||_{L^1}+\epsilon} + \frac{|\boldsymbol{F}^-_{i,:}|}{||\boldsymbol{F}^-_{i,:}||_{L^1}+\epsilon}\right) \tag{17}$$

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

$$\boldsymbol{B}_{dx-0pad}(\boldsymbol{F})_{i,:} = \begin{cases} \boldsymbol{F}'^+_{i,:} & \text{if } \sum_j \boldsymbol{F}'^-_{i,j} = 0 \\ \boldsymbol{F}'^-_{i,:} & \text{if } \sum_j \boldsymbol{F}'^+_{i,j} = 0 \\ \frac{1}{2}\left(\boldsymbol{F}'^+_{i,:} + \boldsymbol{F}'^-_{i,:} - \text{diag}\left(\sum_j \boldsymbol{F}'^+_{:,j} + \boldsymbol{F}'^-_{:,j}\right)_{i,:}\right), & \text{otherwise} \end{cases} \tag{18}$$

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

A.7. Extending the radius of the aggregation kernel

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 $\boldsymbol{F}$.

$$\boldsymbol{B}(\boldsymbol{F}) = \boldsymbol{B}^+(\boldsymbol{F}) - \boldsymbol{B}^-(\boldsymbol{F}) \tag{19}$$

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 20 for any graph G.

Definition 4 (General radius R n-directional kernel)**.** *Let S_n be the group of permutations over n elements with a set of directional fields $\boldsymbol{F}_i$.*

$$\boldsymbol{B}_R := \underbrace{\sum_{\substack{V=\{v_1,v_2,...,v_n\}\in\mathbb{N}^n \\ ||V||_{L^1}\leq R, \quad -R\leq v_i\leq R}}}_{\substack{\text{Any choice of walk } V \text{ with at most } R \text{ steps} \\ \text{using all combinations of } v_1, v_2, ..., v_n}} \underbrace{\sum_{\sigma\in S_n}}_{\substack{\text{optional} \\ \text{permutations}}} a_V \underbrace{\prod_{j=1}^{N} (\boldsymbol{B}_{fb}^{sgn(v_{\sigma(j)})}(\boldsymbol{F}_{\sigma(j)}))^{|v_{\sigma(j)}|}}_{\text{Aggregator following the steps } V, \text{ permuted by } S_n} \tag{20}$$

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 $\sum_{r=0}^{R} \sum_{l=1}^{min(n,r)} 2^r \binom{n}{l} \binom{r-1}{l-1}$.

A.8. Arcsine of the eigenvectors

Since the eigenvectors $\boldsymbol{\phi}_i$ are equivalent to the Fourier basis and represent the waves in the graphs, then it is expected that they behave similarity to sine/cosine waves when the graph is similar to a grid. This is further highlighted by the proof that the eigenvectors of a grid are all sines/cosines in appendix D.4.

Hence, when we define the field $\boldsymbol{F}$ as $\boldsymbol{F}^i = \nabla \boldsymbol{\phi}_i$, we must realize that the gradient will be lower near the minima/maxima of the eigenvector, as it is the case with sine/cosine waves. In the paper, we cope with this problem by dividing by the norm of the field $\|\boldsymbol{F}\|_{L^1}$ in equations 5 and 6.

Another solution is to use the arcsine of the eigenvectors so that the function eigenvectors become similar to triangle functions and the gradient is almost uniform. However, since the arcsine function works only in the range $[-1, 1]$, then we must first normalize the eigenvector by it's maximum, as given by equation 21.

$$\boldsymbol{F}^i_{\text{asin}} = \nabla \arcsin\left(\frac{\boldsymbol{\phi}_i}{\max(|\boldsymbol{\phi}_i|)}\right) \tag{21}$$

B. Appendix - Data augmentation

B.1. Generalizing image augmentation to graphs

The simplest augmentation is the vector field flipping, which is done changing the sign of the field $\boldsymbol{F}$, as stated in definition 5. This changes the sign of $\boldsymbol{B}_{dx}$, but leaves $\boldsymbol{B}_{av}$ unchanged.

Definition 5 (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 fields in a graph, with $\hat{\boldsymbol{F}}_1$ and $\hat{\boldsymbol{F}}_2$ being the field normalized such that each row has a unitary L^2-norm. Define the angle vector $\boldsymbol{\alpha}$ by $\langle (\hat{\boldsymbol{F}}_1)_{i,:}, (\hat{\boldsymbol{F}}_2)_{i,:} \rangle = \cos(\boldsymbol{\alpha}_i)$. The vector field $\hat{\boldsymbol{F}}_2^{\perp}$ is the normalized component of $\hat{\boldsymbol{F}}_2$ perpendicular to $\hat{\boldsymbol{F}}_1$. The equation below defines $\hat{\boldsymbol{F}}_2^{\perp}$. The next equation defines the angle

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

Notice that we then have the decomposition $(\hat{\boldsymbol{F}}_2)_{i,:} = \cos(\boldsymbol{\alpha}_i)(\hat{\boldsymbol{F}}_1)_{i,:} + \sin(\boldsymbol{\alpha}_i)(\hat{\boldsymbol{F}}_2^{\perp})_{i,:}$.

Definition 6 (Rotation of the vector fields)**.** *For $\hat{\boldsymbol{F}}_1$ and $\hat{\boldsymbol{F}}_2$ non-colinear vector fields with each vector of unitary length, their rotation by the angle θ in the plane formed by $\{\hat{\boldsymbol{F}}_1, \hat{\boldsymbol{F}}_2\}$ is*

$$\begin{aligned} \hat{\boldsymbol{F}}_1^{\theta} &= \hat{\boldsymbol{F}}_1 \text{diag}(\cos\theta) + \hat{\boldsymbol{F}}_2^{\perp} \text{diag}(\sin\theta) \\ \hat{\boldsymbol{F}}_2^{\theta} &= \hat{\boldsymbol{F}}_1 \text{diag}(\cos(\theta + \boldsymbol{\alpha})) + \hat{\boldsymbol{F}}_2^{\perp} \text{diag}(\sin(\theta + \boldsymbol{\alpha})) \end{aligned} \tag{22}$$

Finally, the following augmentation has a similar effect to a wave distortion applied on images.

Definition 7 (Random distortion of the vector field)**.** *For vector field $\boldsymbol{F}$ and anti-symmetric random noise matrix $\boldsymbol{R}$, its randomly distorted field is $\boldsymbol{F}' = \boldsymbol{F} + \boldsymbol{R} \circ \boldsymbol{A}$.*

B.2. Preliminary results of data augmentation

To evaluate the effectiveness of the proposed augmentation, we trained the models on a reduced version of the CIFAR10 dataset. The results in figure 7 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 a reduced test-set performance, but we observe that randomly rotating or distorting the kernels counteracts the overfitting and improves the generalization.

As expected, the performance decreases when the rotation or distortion is too high since the augmented graph changes too much. In computer vision images similar to CIFAR10 are usually rotated by less than $30°$ (Shorten & Khoshgoftaar, 2019; O'Gara & McGuinness, 2019). Further, due to the constant number of parameters across models, less parameters are attributed to the mean aggregation in the directional models, thus it cannot fit well the data when the rotation/distortion is too strong since the directions are less informative. We expect large models to perform better at high angles.

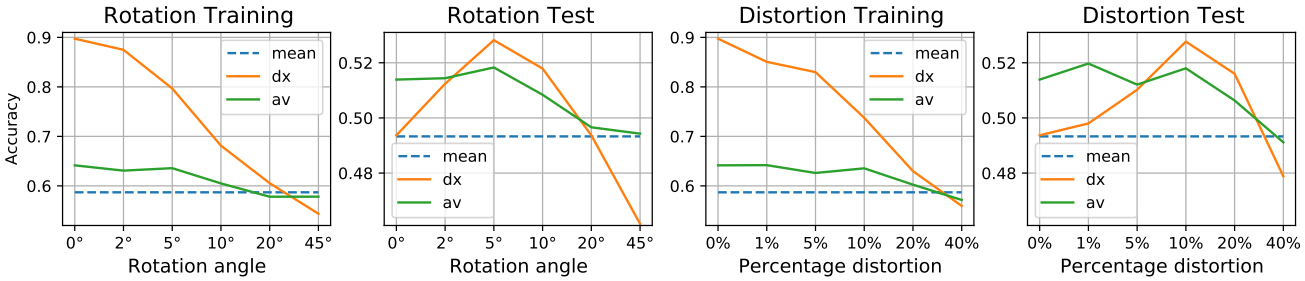


Figure 7. Accuracy of the various models using data augmentation with a *complex* architecture of $\sim 100k$ parameters and trained on 10% of the CIFAR10 training set (4.5k images). An angle of x corresponds to a rotation of the kernel by a random angle sampled uniformly in $(-x°, x°)$ using definition 6 with $\boldsymbol{F}_{1,2}$ being the gradient of the horizontal/vertical coordinates. A noise of $100x\%$ corresponds to a distortion of each eigenvector with a random noise uniformly sampled in $(-x \cdot m, x \cdot m)$ where m is the average absolute value of the eigenvector's components. The *mean* baseline model is not affected by the augmentation since it does not use the underlining vector field.

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[2]).

For the results in figure 6, 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 $\in \{(mean, av_1, av_2), (mean, dx_1, dx_2), (mean, dx_1, dx_2, av_1, av_2), (mean, max, min, dx_1, dx_2), (mean, max, min, av_1, av_2)\}$
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 D.1, 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.

To evaluate the difficulty of computing the eigenvectors on very large graphs, we decided to load the COLLAB dataset comprising of a single graph with 235k nodes and 2.35M edges (Dwivedi et al., 2020). Computing it's first 6 eigenvectors using the scipy *eigsh* function with machine precision took 25.5 minutes on an Intel® Xeon® CPU @ 2.20GHz. This is acceptable, knowing that a general training time can take hours, and that the result can be cached and reused during debugging and hyper-parameter optimization.

[2]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 6

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.

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%

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. In section 2.5, 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 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 - Mathematical proofs

D.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.

□

D.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.

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

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

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

□

D.3. Proof of theorem 2.3 (Gradient steps reduce diffusion distance)

Let x, y be nodes such that $\boldsymbol{\phi}_1(x) < \boldsymbol{\phi}_1(y)$. Let x' be the node obtained from x by taking one step in the direction of $\nabla\boldsymbol{\phi}_1$, then there is a constant C such that for $C \leq t$ we have

$$d_t(x', y) < d_t(x, y).$$

With the reduction in distance being proportional to $e^{-\lambda_1}$.

Recall that $p_k(x, y) = (D^{-1}A)^k_{x,y}$ is the discrete heat kernel at step k, $q_t(x, y) = \sum_{k\geq 0} \frac{e^{-t}t^k}{k!} p_k(x, y)$ is the continuous heat kernel at time t. In (Barlow, 2017), it is shown that the continuous heat kernel is computed by $q_t(x, y) = e^{-t\boldsymbol{L}_{\text{norm}}}$. Following (Coifman & Lafon, 2006) we can diagonalise q_t to get the identity

$$d_t(x, y) = \left(\sum_{i=1}^{n-1} e^{-2t\lambda_i}\Big(\boldsymbol{\phi}_i(x) - \boldsymbol{\phi}_i(y)\Big)^2\right)^{\frac{1}{2}} \tag{23}$$

The inequality $d_t(x', y) < d_t(x, y)$ is equivalent to

$$\sum_{i=2}^{n-1} e^{-2t\lambda_i}\left(\Big(\boldsymbol{\phi}_i(x') - \boldsymbol{\phi}_i(y)\Big)^2 - \Big(\boldsymbol{\phi}_i(x) - \boldsymbol{\phi}_i(y)\Big)^2\right) < e^{-2t\lambda_1}\left(\Big(\boldsymbol{\phi}_1(x) - \boldsymbol{\phi}_1(y)\Big)^2 - \Big(\boldsymbol{\phi}_1(x') - \boldsymbol{\phi}_1(y)\Big)^2\right) \tag{24}$$

The term on the left is bounded above by

$$\sum_{i=2}^{n-1} e^{-2t\lambda_i}\left|\Big(\boldsymbol{\phi}_i(x') - \boldsymbol{\phi}_i(y)\Big)^2 - \Big(\boldsymbol{\phi}_i(x) - \boldsymbol{\phi}_i(y)\Big)^2\right|$$

and this last term is in turn bounded above by

$$e^{-2t\lambda_2}\sum_{i=2}^{n-1}\left|\Big(\boldsymbol{\phi}_i(x') - \boldsymbol{\phi}_i(y)\Big)^2 - \Big(\boldsymbol{\phi}_i(x) - \boldsymbol{\phi}_i(y)\Big)^2\right|$$

Inequality 24 will then hold if

$$e^{-2t\lambda_2}\sum_{i=2}^{n-1}\left|\Big(\boldsymbol{\phi}_i(x')-\boldsymbol{\phi}_i(y)\Big)^2-\Big(\boldsymbol{\phi}_i(x)-\boldsymbol{\phi}_i(y)\Big)^2\right|<e^{-2t\lambda_1}\left(\Big(\boldsymbol{\phi}_1(x)-\boldsymbol{\phi}_1(y)\Big)^2-\Big(\boldsymbol{\phi}_1(x')-\boldsymbol{\phi}_1(y)\Big)^2\right)$$

and this is equivalent to

$$\frac{1}{2(\lambda_1-\lambda_2)}\log\left(\frac{\left(\Big(\boldsymbol{\phi}_1(x)-\boldsymbol{\phi}_1(y)\Big)^2-\Big(\boldsymbol{\phi}_1(x')-\boldsymbol{\phi}_1(y)\Big)^2\right)}{\sum_{i=2}^{n-1}\left|\Big(\boldsymbol{\phi}_i(x')-\boldsymbol{\phi}_i(y)\Big)^2-\Big(\boldsymbol{\phi}_i(x)-\boldsymbol{\phi}_i(y)\Big)^2\right|}\right)<t$$

if we take t to be larger than the term on the left the inequality we get $d_t(x',y)<d_t(x,y)$.

The constant C in the statement is the constant on the left side of the inequality. It is also interesting to note that C is expected to be positive since the term $\lambda_1-\lambda_2$ is negative and the argument of the $\log$ will most likely be <1.

D.4. Proof for Lemma D.1 (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 D.1 (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

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

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. $2-2\cos\left(\frac{\pi}{N_i}\right)$. □

D.5. 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 D.2. *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 D.3 (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 D.1. 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 D.2 obtained by adding n 1-dimensional kernels, with each kernel being in a different axis of the grid as per Lemma D.1. See figure 4 for a visual example in 2D.

□

D.6. 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

$$\boldsymbol{a}_{5\times5} = \begin{pmatrix} 0 & 0 & a_{-2,0} & 0 & 0 \\ 0 & a_{-1,-1} & a_{-1,0} & a_{-1,1} & 0 \\ a_{0,-2} & a_{0,-1} & a_{0,0} & a_{0,1} & a_{0,2} \\ 0 & a_{1,-1} & a_{1,0} & a_{1,1} & 0 \\ 0 & 0 & a_{2,0} & 0 & 0 \end{pmatrix}$$

since we supposed the $N_1 \times N_2$ grid was such that $N_1 > N_2$, by theorem D.1, we have that $\boldsymbol{\phi}_1$ is depending only in the first variable x_1 and is monotone in x_1. Recall from D.1 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 D.1 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 D.3. 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 D.1 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 $\boldsymbol{F}_4$.

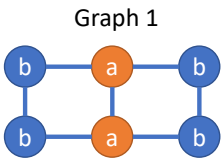
□

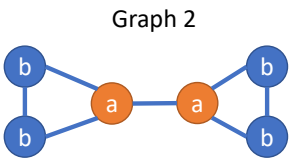
D.7. Proof for theorem 2.5 (Comparison with 1-WL test)

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

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.

□