Graph Neural Networks Are More Powerful Than we Think

Anonymous authors
Paper under double-blind review

Abstract

Graph Neural Networks (GNNs) are powerful convolutional architectures that have shown remarkable performance in various node-level and graph-level tasks. Despite their success, the common belief is that the expressive power of standard GNNs is limited and that they are at most as discriminative as the Weisfeiler-Lehman (WL) algorithm. In this paper we argue the opposite and show that the WL algorithm is the upper bound only when the input to the GNN is the vector of all ones. In this direction, we derive an alternative analysis that employs linear algebraic tools and characterize the representational power of GNNs with respect to the eigenvalue decomposition of the graph operators. We show that GNNs can distinguish between any graphs that differ in at least one eigenvalue and design simple GNN architectures that are provably more expressive than the WL algorithm. Thorough experimental analysis on graph isomorphism and graph classification datasets corroborates our theoretical results and demonstrates the effectiveness of the proposed architectures.

1 Introduction

Graph Neural Networks (GNNs) have emerged in the field of machine learning and artificial intelligence as powerful tools that process network structures and network data. Their convolutional architecture allows them to inherit all the favorable properties of convolutional neural networks (CNNs), while they also exploit the graph structure. Biology (Gainza et al., 2020; Strokach et al., 2020; Jiang et al., 2021), quantum chemistry (Gilmer et al., 2017), as well as robotics (Lima et al., 2020; Cranmer et al., 2021), social networks and recommender systems (Ying et al., 2018; Wu et al.) are typical fields where GNNs have been applied. GNNs have demonstrated state-of-the-art performance in various downstream tasks, associated with these fields, that include (but are not limited to) node and graph classification, link prediction and network regression.

Despite their remarkable performance, the success of GNNs is still to be demystified. A lot of research has been conducted to theoretically support the experimental developments, focusing on understanding the functionality of GNNs and analyzing their properties. In particular, permutation invariance-equivariance (Maron et al., 2018), stability to perturbations (Gama et al., 2020) and transferability (Ruiz et al., 2020a; Levie et al., 2021) are properties tantamount to the success of the GNNs. Lately, the research focus has been shifted towards analyzing their expressive power, since the universality of GNNs depends on their ability to produce different outputs for different graphs. The common belief is that standard GNNs have limited expressive power (Xu et al., 2019) and that it is upper bounded by the expressive power of the Weisfeiler-Lehman (WL) algorithm (Weisfeiler & Leman, 1968). This induced increased research activity towards building complex and more expressive GNNs. In this work we argue the opposite. We prove that standard graph convolutional structures are capable of distinguishing between graphs that the WL algorithm cannot and therefore complex GNNs are not necessary to break the WL limits.

Our work is motivated by the following questions. *How expressive are GNNs? Can simple convolutional architectures be more expressive than the WL algorithm?* The answer to both questions is definitive. Our analysis utilizes spectral decomposition tools to show that the source of the WL test as a limit for the expressive power of GNNs is the use of the all-one vector as an input. Our spectral analyses corroborate that, indeed, if a GNN is initialized with the all-one vector as an input, the WL test is a limit on the expressive power of GNNs. However, if we initialize a GNN with white random inputs, it is possible to discriminate, at least, any pair of graphs with at least one different eigenvalue. This implies that standard GNNs are provably more expressive than the WL algorithm as they discriminate between graphs that fail the WL test, yet have different eigenvalues. In fact, having at least one different eigenvalue is a very mild condition that is rarely not met in practice.

Using white noise as an input to a GNN may be computationally costly. We show, however, that there are two alternative GNN architectures that are equivalent to a GNN architecture with white random inputs: (i) A GNN that operates on matrix representations of the graph without requiring any input. (ii) A GNN in which input features are derived from powers of matrix representations of the graph. Our numerical results show that our proposed GNNs are better discriminator in some graph classification problems.

Our contribution is summarized as follows:

(C1) We characterize the expressive power of GNNs employing spectral decomposition tools.

(C2) We explain that the WL algorithm is a limit on the expressive power of GNNs only when we use the all-one vector as an input.

(C3) We show that standard GNNs can distinguish between any pair of graphs with at least one different eigenvalue if node features are initialized with white random noise. This implies that standard GNNs are provably more expressive than the WL algorithm.

(C4) We design equivalent architectures that circumvent the use of random input features. These architectures can use features derived from powers of matrix representations or can avoid the use of features altogether.

(C5) We demonstrate the effectiveness of using GNNs with white random inputs, or the proposed alternatives, in graph isomorphism and graph classification datasets.

Related work: The first work to study the approximation properties of the GNNs was by (Scarselli et al., 2008a). Along the same lines (Maron et al., 2019b; Keriven & Peyré, 2019) discuss the universality of GNNs for permutation invariant or equivariant functions. Then the scientific attention focused on the ability of GNNs to distinguish between non-isomorphic graphs. The works of (Morris et al., 2019; Xu et al., 2019) place the expressive power of GNNs with respect to that of the WL algorithm and prompted various follow-up works in the area. Specifically, (Abboud et al., 2021; Sato et al., 2021) use random features to increase the separation capabilities of GNNs, whereas (Tahmasebi et al., 2020) produce features related to the subgraph information by adding a neighborhood pooling layer. Furthermore (Corso et al., 2020; Beaini et al., 2021) use multiple and directional aggregators, respectively, to increase the GNN expressivity. GNNs that use k-tuple and k-subgraph information have been designed by (Maron et al., 2019a; Murphy et al., 2019; Azizian et al., 2020; Morris et al., 2020; Geerts & Reutter, 2021). These works use a tensor framework, and employ more expressive structures compared to simple GNNs. However, they are usually computationally heavier to implement and also prone to overfitting. Moreover, (Balcilar et al., 2021) design convolutions in the spectral domain to produce powerful GNNs, whereas (Loukas, 2019) studies the learning capabilities of a GNN with respect to its width and depth. Finally, (Chen et al., 2019) reveal a connection between the universal approximation and the capacity capabilities of GNNs.

2 On the expressive power of GNNs

Studying the expressive power of GNNs has attracted significant attention, since it sheds light on the success and general functionality of graph convolutional architectures. One of the most influential works by (Xu et al., 2019) compares the representational capabilities of GNNs with those of the WL algorithm (color refinement algorithm). The claim is that GNNs are at most as powerful as the WL algorithm in distinguishing between different graphs. This is indeed true when the input to the GNN is the vector of all ones, i.e., $\boldsymbol{x} = \mathbf{1}$ and therefore the propagated graph signals are of the form $\boldsymbol{S}^k\mathbf{1}$, where $\boldsymbol{S} \in \{0,1\}^{N\times N}$ is the graph adjacency and $\boldsymbol{S}^k$ is the k-th power of $\boldsymbol{S}$.

A question that naturally arises is *'Why limit attention to input features $\boldsymbol{x} = \mathbf{1}$?'*. On the one hand, various systems of practical interest provide access to a set of attributes or graph signals, $\boldsymbol{X} \in \mathbb{R}^{N\times D}$, that are associated with each node. These attributes contain rich information beyond the connectivity provided by the graph, as opposed to $\boldsymbol{x} = \boldsymbol{1}$ inputs. On the other hand, when attributes are not available, we can design artificial features from the graph that incorporate valuable knowledge, not captured by $\boldsymbol{S}^k\boldsymbol{1}$. The need for further analysis with general input signals is therefore clear.

To better understand the expressive power of GNNs we study general inputs. Consider then graphs $\mathcal{G},\ \hat{\mathcal{G}}$ with graph operators $\boldsymbol{S},\ \hat{\boldsymbol{S}}$. In this paper $\boldsymbol{S},\ \hat{\boldsymbol{S}}$ denote the graph adjacencies, but other choices of graph operators can be used, e.g., graph Laplacians or weighted graph adjacencies/Laplacians. We assume that $\boldsymbol{S},\ \hat{\boldsymbol{S}}$ are both symmetric and thus admit eigenvalue decompositions as:

$$\boldsymbol{S} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T, \quad \hat{\boldsymbol{S}} = \hat{\boldsymbol{U}}\hat{\boldsymbol{\Lambda}}\hat{\boldsymbol{U}}^T, \tag{1}$$

The equations in (1) represent the spectral decompositions of $\mathcal{G}$ and $\hat{\mathcal{G}}$, where $\boldsymbol{U},\ \hat{\boldsymbol{U}}$ are orthogonal matrices containing the eigenvectors and $\boldsymbol{\Lambda},\ \hat{\boldsymbol{\Lambda}}$ are the diagonal matrices of corresponding eigenvalues. The graphs $\mathcal{G},\ \hat{\mathcal{G}}$ are non-isomorphic if and only if there does not exist a permutation matrix $\boldsymbol{\Pi}$ such that $\boldsymbol{S} = \boldsymbol{\Pi}\hat{\boldsymbol{S}}\boldsymbol{\Pi}^T$. A broad class of non-isomorphic graphs have at least one different eigenvalue. To be more precise, let $\mathcal{S} = \{\lambda_1, \dots, \lambda_N\}$ be the multiset containing the eigenvalues of $\boldsymbol{S}$ and $\hat{\mathcal{S}} = \{\hat{\lambda}_1, \dots, \hat{\lambda}_N\}$ be the multiset containing the eigenvalues of $\hat{\boldsymbol{S}}$. The following assumption is heavily used in the main part of this paper:

Assumption 2.1 *$\boldsymbol{S},\ \hat{\boldsymbol{S}}$ have at least one different eigenvalue, i.e., there exists μ_k with multiplicity m and corresponding eigenspace $\boldsymbol{V} \in \mathbb{R}^{N\times m}$, such that $\mu_k \in \mathcal{S}$ but $\mu_k \notin \hat{\mathcal{S}}$.*

When Assumption 2.1 holds, $\mathcal{G},\ \hat{\mathcal{G}}$ are always non-isomorphic. Assumption 2.1 is not restrictive. Real non-isomorphic graphs have different eigenvalues with very high probability (Haemers & Spence, 2004). Corner cases where Assumption 2.1 doesn't hold are studied in Appendix H.

First, we consider GNNs that are constructed by the following modules, corresponding to the neurons of a typical (non-graph) neural network:

$$\boldsymbol{Y} = \sigma\left(\sum_{k=0}^{K-1} \boldsymbol{S}^k\boldsymbol{X}\boldsymbol{H}_k\right). \tag{2}$$

The module in (2) is composed by a graph filter of length K followed by a nonlinearity $\sigma(\cdot)$. $\boldsymbol{H}_k$ represents the filter parameters and can be a matrix, a vector, or a scalar. In order to characterize the representational power of GNNs with general input, we provide the following theorem:

Theorem 2.2 *Let $\mathcal{G},\ \hat{\mathcal{G}}$ be non-isomorphic graphs with graph signals $\boldsymbol{X},\ \hat{\boldsymbol{X}}$. There exist a GNN that tells $\mathcal{G}$ and $\hat{\mathcal{G}}$ apart if:*

1. *There does not exist permutation matrix $\boldsymbol{\Pi}$ such that $\boldsymbol{X} = \boldsymbol{\Pi}\hat{\boldsymbol{X}}$, or*
2. *Assumption 2.1 holds and $\boldsymbol{V}^T\boldsymbol{X} \neq \boldsymbol{0}$.*

Theorem 2.2 highlights the importance of the input $\boldsymbol{X}$ in the representational capabilities of a GNN. For problems in which inputs are given, it states that a GNN can distinguish between non-isomorphic graphs if they have different graph signals or their signals are not orthogonal to the eigenspace associated with the eigenvalue that differentiates them. In problems where inputs are not available, Theorem 2.2 provides guidelines on how to design input $\boldsymbol{X}$ from the graph.

Theorem 2.2 also indicates that the limitations of GNNs discussed in (Xu et al., 2019) are not due to the architecture but they are limitations associated with the input. In particular, $\boldsymbol{x} = \boldsymbol{1}$ fails to satisfy condition 1, while it is also prone to fail condition 2, since the majority of real graphs have eigenvectors that are orthogonal to $\boldsymbol{1}$. Thus, the challenge lies in designing GNN inputs that satisfy conditions 1 and 2. This is accomplished in section 5, where we propose to construct $\boldsymbol{X}$ as:

$$\boldsymbol{X} = \left[\text{diag}\left(\boldsymbol{S}^0\right), \text{diag}\left(\boldsymbol{S}^1\right), \text{diag}\left(\boldsymbol{S}^2\right), \dots, \text{diag}\left(\boldsymbol{S}^{D-1}\right)\right], \tag{3}$$

where diag $\left(\boldsymbol{S}^k\right)$ refers to vector containing the diagonal entries of matrix $\boldsymbol{S}^k$. Proper choice of D guarantees that the graph signal $\boldsymbol{X}$ in (3) satisfies both conditions of Theorem 2.2 and enables GNNs to distinguish between any non-isomorphic graphs that have at least one different eigenvalue. A nice interpretation of this result is given in section 5 and shows that $\boldsymbol{X}$ in (3) combines information from both the k-hop degrees ($\boldsymbol{S}^k\mathbf{1}$) and the high-order subgraphs that appear in the network. As shown in the next section, the WL algorithm cannot always tell graphs with different eigenvalues apart, which implies that GNNs are more expressive than the WL algorithm for this class of graphs.

3 Limitations of GNNs with $\boldsymbol{x} = \mathbf{1}$ input and the WL algorithm

Using Theorem 2.2 we can explain why feeding a GNN with $\boldsymbol{x} = \mathbf{1}$ is limiting. The limitations associated with input $\boldsymbol{x} = \mathbf{1}$ are also highly related to the limitations of the WL algorithm. The problem appears in graphs that admit a spectral decomposition with eigenvectors that are orthogonal to $\mathbf{1}$ (they sum up to zero). Following condition 2 in Theorem 2.2, if two graphs are the same except eigenvalues that correspond to eigenvectors that sum up to zero, then GNNs with input $\boldsymbol{x} = \mathbf{1}$ will fail to tell the two graphs apart. To see this consider the graphs $\mathcal{G}$, $\hat{\mathcal{G}}$ with spectral decompositions:

$$\boldsymbol{S} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T = \lambda_1\boldsymbol{u}_1\boldsymbol{u}_1^T + \lambda_2\boldsymbol{u}_2\boldsymbol{u}_2^T + \lambda_3\boldsymbol{u}_3\boldsymbol{u}_3^T, \tag{4}$$

$$\hat{\boldsymbol{S}} = \hat{\boldsymbol{U}}\hat{\boldsymbol{\Lambda}}\hat{\boldsymbol{U}}^T = \lambda_1\boldsymbol{u}_1\boldsymbol{u}_1^T + \lambda_2\boldsymbol{u}_2\boldsymbol{u}_2^T + \hat{\lambda}_3\boldsymbol{u}_3\boldsymbol{u}_3^T, \tag{5}$$

where $\lambda_3 \neq \hat{\lambda}_3$. If $\boldsymbol{u}_3$ is orthogonal to $\mathbf{1}$ then:

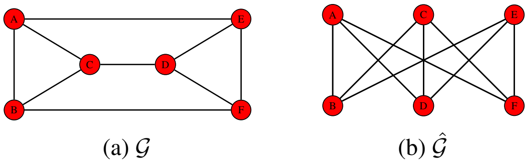
$$\boldsymbol{S}^k\mathbf{1} = \boldsymbol{U}\boldsymbol{\Lambda}^k\boldsymbol{U}^T\mathbf{1} = \lambda_1^k\boldsymbol{u}_1\boldsymbol{u}_1^T\mathbf{1} + \lambda_2^k\boldsymbol{u}_2\boldsymbol{u}_2^T\mathbf{1} + \lambda_3^k\boldsymbol{u}_3\boldsymbol{u}_3^T\mathbf{1} = \lambda_1^k\left(\boldsymbol{u}_1^T\mathbf{1}\right)\boldsymbol{u}_1 + \lambda_2^k\left(\boldsymbol{u}_2^T\mathbf{1}\right)\boldsymbol{u}_2 \tag{6}$$

$$\hat{\boldsymbol{S}}^k\mathbf{1} = \hat{\boldsymbol{U}}\hat{\boldsymbol{\Lambda}}^k\hat{\boldsymbol{U}}^T\mathbf{1} = \lambda_1^k\boldsymbol{u}_1\boldsymbol{u}_1^T\mathbf{1} + \lambda_2^k\boldsymbol{u}_2\boldsymbol{u}_2^T\mathbf{1} + \hat{\lambda}_3^k\boldsymbol{u}_3\boldsymbol{u}_3^T\mathbf{1} = \lambda_1^k\left(\boldsymbol{u}_1^T\mathbf{1}\right)\boldsymbol{u}_1 + \lambda_2^k\left(\boldsymbol{u}_2^T\mathbf{1}\right)\boldsymbol{u}_2 \tag{7}$$

The diffused information in GNNs with this naive input is of the form $\boldsymbol{S}^k\mathbf{1}$ and therefore in the above example the decisive information that differentiates the two graphs is omitted.

Graphs with eigenvectors orthogonal to $\mathbf{1}$ can also affect the performance of the WL algorithm. In the absence of features the WL algorithm is initialized with $\boldsymbol{x} = \boldsymbol{S}\mathbf{1}$, which is propagated through the nodes iteratively. In graphs with eigenvectors orthogonal to $\mathbf{1}$, the propagated degrees have suffered critical information loss in the initialization, which in certain graph structures is impossible to recover, as WL iterations progress. Further analysis on this subject can be found in Appendix C.

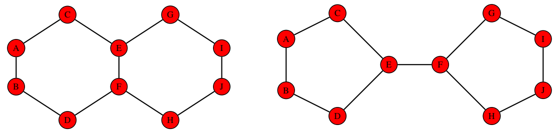
Classic examples of graphs with different eigenvalues, that the WL algorithm and GNNs with $\boldsymbol{x} = \mathbf{1}$ input cannot tell apart, are presented in Figs. 1, 2. In particular, these approaches decide that $\mathcal{G}$ and $\hat{\mathcal{G}}$ in Fig. 1 and $\mathcal{G}$ and $\hat{\mathcal{G}}$ in Fig. 2 are the same. This is due to the fact that these graphs contain eigenvectors that are orthogonal to $\mathbf{1}$. The case of Fig. 1 is straightforward. All the nodes of $\mathcal{G}$ and $\hat{\mathcal{G}}$ have the same degree, i.e., $\boldsymbol{x} = \mathbf{1}$ is an eigenvector in both graphs and thus orthogonal to all the remaining eigenvectors. As a result, the node degrees (which are the same for both graphs) are the only information that the WL algorithm and GNNs with $\mathbf{1}$ input are able to process. The case of Fig. 2 is more complicated; $\boldsymbol{x} = \mathbf{1}$ is not an eigenvector in any of the graphs, but it is orthogonal to the eigenvectors corresponding to the eigenvalues that differentiate the two graphs. Consequently, the operation $\boldsymbol{S}\mathbf{1}$ negates vital information and the two approaches fail.



(a) $\mathcal{G}$ (b) $\hat{\mathcal{G}}$

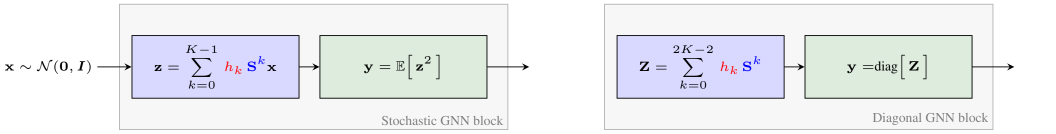
Figure 1: WL indistinguishable graphs.

Detailed information about the eigenvalues and eigenvectors of the graphs in Figs. 1, 2 can be found in Tables 7, 8 of Appendix K. This information corroborates the issues discussed in the previous paragraph. As noted earlier and will be explained in more detail in the upcoming sections, carefully designed GNNs overcome these issues and decide that $\mathcal{G}$ and $\hat{\mathcal{G}}$ in both Figs. 1, 2 are non-isomorphic.



(a) $\mathcal{G}$ (b) $\hat{\mathcal{G}}$

Figure 2: WL indistinguishable graphs


$\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
$\mathbf{z} = \sum_{k=0}^{K-1} h_k \mathbf{S}^k \mathbf{x}$
$\mathbf{y} = \mathbb{E}\left[\mathbf{z}^2\right]$
Stochastic GNN block
$\mathbf{Z} = \sum_{k=0}^{2K-2} h_k \mathbf{S}^k$
$\mathbf{y} = \mathrm{diag}\left[\mathbf{Z}\right]$
Diagonal GNN block

(a) Stochastic GNN module

(b) equivalent model

Figure 3: GNN with random Gaussian input

4 Feeding the GNN with random input

In this section we overcome the GNN limitations associated with $\boldsymbol{x} = \mathbf{1}$ by feeding a GNN with white Gaussian input. We consider again the GNN module in (2) where $\boldsymbol{H}_k$ is a scalar, i.e., $\boldsymbol{y} = \sigma\left(\sum_{k=0}^{K-1} h_k \boldsymbol{S}^k \boldsymbol{x}\right)$. Before choosing an appropriate nonlinearity, let us focus on the linear convolutional graph filter of length K:

$$\boldsymbol{z} = \sum_{k=0}^{K-1} h_k \boldsymbol{S}^k \boldsymbol{x}, \tag{8}$$

which we load with random input $\boldsymbol{x} \in \mathbb{R}^N$ that is drawn from a Gaussian distribution, i.e., $\boldsymbol{x} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. Since $\boldsymbol{x}$ is a random vector with $\mathbb{E}\left[\boldsymbol{x}\right] = \mathbf{0}$, $\boldsymbol{z}$ is also a random vector with $\mathbb{E}\left[\boldsymbol{z}\right] = \mathbf{0}$. Thus, the expected value provides no information about the network. Measuring the covariance, on the other hand, yields:

$$\begin{aligned}\text{cov}\left[\boldsymbol{z}\right] = \mathbb{E}\left[\boldsymbol{z}\boldsymbol{z}^T\right] &= \mathbb{E}\left[\sum_{k=0}^{K-1} h_k \boldsymbol{S}^k \boldsymbol{x}\boldsymbol{x}^T \sum_{m=0}^{K-1} h_m {\boldsymbol{S}^m}^T\right] = \sum_{k=0}^{K-1} h_k \boldsymbol{S}^k \mathbb{E}\left[\boldsymbol{x}\boldsymbol{x}^T\right] \sum_{m=0}^{K-1} h_m \boldsymbol{S}^m \\ &= \sum_{k=0}^{K-1} h_k \boldsymbol{S}^k \sum_{m=0}^{K-1} h_m \boldsymbol{S}^m = \sum_{k=0}^{K-1}\sum_{m=0}^{K-1} h_k h_m \boldsymbol{S}^k \boldsymbol{S}^m = \sum_{k=0}^{2K-2} h'_k \boldsymbol{S}^k,\end{aligned} \tag{9}$$

where $h'_k = \sum_{m,l} h_m h_l,$ such that $m + l = k$. The results of equation (9) are noteworthy. We have shown that the covariance of a graph filter with random uncorrelated input corresponds to a different graph filter with no input. Furthermore, the resulting filter has length $2K - 1$, whereas the original filter has length K. In other words the nonlinearity introduced by the covariance computation enables the filter to gather information from a broader neighborhood compared to the initial filter. However, there is a caveat that the degrees of freedom for h' are K and not $2K - 1$. Further discussion on the subject can be found in Appendix D.

In practice we want to associate the output of a GNN with a feature for each node that is permutation equivariant. This is not the case with the rows or columns of the covariance matrix in (9). Therefore we choose $\sigma(\cdot)$ to be the variance of each node i.e.,

$$\boldsymbol{y} = \sigma\left(\boldsymbol{z}\right) = \text{var}\left[\boldsymbol{z}\right] = \mathbb{E}\left[\boldsymbol{z}^2\right] = \text{diag}\left(\text{cov}\left[\boldsymbol{z}\right]\right) = \text{diag}\left(\sum_{k=0}^{2K-2} h'_k \boldsymbol{S}^k\right) = \sum_{k=0}^{2K-2} h'_k \text{diag}\left(\boldsymbol{S}^k\right). \tag{10}$$

The proposed stochastic GNN module is illustrated in Fig. 3a. Regarding its expressive power, we present the following theorem:

Theorem 4.1 *Let $\mathcal{G}$, $\hat{\mathcal{G}}$ be non-isomorphic graphs. If Assumption 2.1 holds, there exists a GNN with modules as in Fig. 3a that tells the two graphs apart.*

In simple words, a GNN with modules as in Fig. 3a can always distinguish between graphs that have at least one different eigenvalue.

Proposition 4.1 *The GNN module in Fig. 3a with random input $\boldsymbol{x} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ is equivalent to the GNN module in Fig. 3b with no input up to degrees of freedom (dependencies) in the filter parameters.*

The proof of Proposition 4.1 is the combination of equations (9), (10). The claim is eminent. It proves equivalence of two GNN architectures; a stochastic graph filter with Gaussian input followed by a variance nonlinearity with a deterministic graph filter followed by a diagonal operator. Depending on the problem and the variance of the system one has the option to choose either of them. Further discussion on the stochastic approach can be found in Appendix D.

5 THE DIAGONAL MODULE

Proposition 4.1 proved the equivalence of the two GNN modules in Fig. 3. In this section we focus on the module in 3a and analyze its unique properties. To be more precise, we study the following diagonal GNN module:

$$\boldsymbol{y} = \sigma\left(\sum_{k=0}^{K-1} h_k \text{diag}\left(\boldsymbol{S}^k\right)\right), \tag{11}$$

Note that the module in (11) is not exactly the same as the one in Fig. 3b, since a nonlineatity is added and the filter is of length K. As an example, we test the proposed diagonal module on the graphs of Figs. 1, 2, and present the output $\boldsymbol{y}$ of (11) with parameters $(h_0, h_1, h_2, h_3, h_4, h_5) = (10, 1, -\frac{1}{2}, \frac{1}{3}, -\frac{1}{4}, \frac{1}{5})$ and ReLU nonlinearity, in Table 1.

Table 1: Outputs $\boldsymbol{y}$ of $\mathcal{G}$ and $\hat{\boldsymbol{y}}$ of $\hat{\mathcal{G}}$ of the proposed diagonal module for the graphs in Figs. 1, 2.

GRAPH		NODE									
		A	B	C	D	E	F	G	H	I	J
FIG. 1	$\boldsymbol{y}$	10.42	10.42	10.42	10.42	10.42	10.42	-	-	-	-
	$\hat{\boldsymbol{y}}$	1.75	1.75	1.75	1.75	1.75	1.75	-	-	-	-
FIG. 2	$\boldsymbol{y}$	7.5	7.5	7.25	7.25	5.25	5.25	7.25	7.25	7.5	7.5
	$\hat{\boldsymbol{y}}$	7.9	7.9	7.65	7.65	5.65	5.65	7.65	7.65	7.9	7.9

We observe that the output (11) of the proposed diagonal module produces embeddings that are different for the nodes of $\mathcal{G}$ and $\hat{\mathcal{G}}$ in both Figs. 1, 2. Therefore, there does not exist permutation matrix $\boldsymbol{\Pi}$ such that $\boldsymbol{y} = \boldsymbol{\Pi}\hat{\boldsymbol{y}}$ and the proposed architecture is able to tell $\mathcal{G}$ and $\hat{\mathcal{G}}$ apart in both Figs. 1, 2. This is in stark contrast to GNNs with $\boldsymbol{x} = \mathbf{1}$ input and the WL algorithm that fail to distinguish between these graphs (as discussed in section 3). The success of the proposed diagonal module lies in the spectral decomposition of (11):

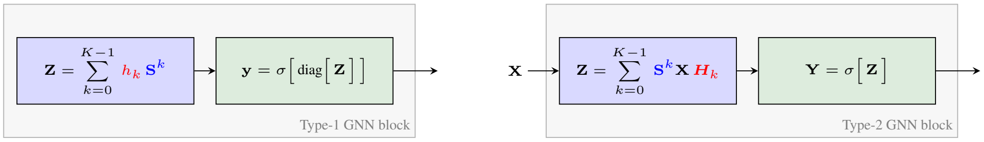
$$\boldsymbol{y} = \sigma\left(\sum_{k=0}^{K-1} h_k \text{diag}\left(\sum_{n=1}^{N} {\lambda_n}^k \boldsymbol{u}_n \boldsymbol{u}_n^T\right)\right) = \sigma\left(\sum_{k=0}^{K-1}\sum_{n=1}^{N} h_k \lambda_n^k |\boldsymbol{u}_n|^2\right). \tag{12}$$

In simple words, the frequency response of the proposed GNN module depends on the absolute values of the graph adjacency eigenvectors. On the contrary, the modules of GNNs, as in (2), admit a different frequency representation when loaded with $\boldsymbol{x} = \mathbf{1}$ input:

$$\boldsymbol{y}_1 = \sigma\left(\sum_{k=0}^{K-1} h_k \boldsymbol{S}^k \mathbf{1}\right) = \sigma\left(\sum_{k=0}^{K-1} h_k \sum_{n=1}^{N} {\lambda_n}^k \boldsymbol{u}_n \boldsymbol{u}_n^T \mathbf{1}\right) = \sigma\left(\sum_{k=0}^{K-1}\sum_{n=1}^{N} h_k {\lambda_n}^k \boldsymbol{u}_n \boldsymbol{u}_n^T \mathbf{1}\right), \tag{13}$$

where $\boldsymbol{y}_1$ denotes the output of a GNN module with $\boldsymbol{x} = \mathbf{1}$ input. As we can see both outputs $\boldsymbol{y},\ \boldsymbol{y}_1$ are functions of the graph eigenvectors. The question that arises is which function, $|\boldsymbol{u}_n|$ or $\left(\boldsymbol{u}_n^T \mathbf{1}\right)\boldsymbol{u}_n$, results in more expressive GNNs. The naive answer is that depending on the graph, there is a trade-off between the information loss caused by $|\boldsymbol{u}_n|$ or $\left(\boldsymbol{u}_n^T \mathbf{1}\right)\boldsymbol{u}_n$. However, after adding a second layer, GNNs with diagonal modules are always more powerful than GNNs initialized by $\mathbf{1}$. This will be explained in more detail in the next section.

Remark 5.1 *A closer look at equations* (11) *and* (12)*, reveals further insights regarding the proposed architecture. In particular, the proposed diagonal module is constructed by the diagonal elements of*


$\mathbf{Z} = \sum_{k=0}^{K-1} h_k \mathbf{S}^k$
$\mathbf{y} = \sigma\left[\mathrm{diag}\left[\mathbf{Z}\right]\right]$
Type-1 GNN block
$\mathbf{X}$
$\mathbf{Z} = \sum_{k=0}^{K-1} \mathbf{S}^k \mathbf{X} \boldsymbol{H}_k$
$\mathbf{Y} = \sigma\left[\mathbf{Z}\right]$
Type-2 GNN block

(a) Type-1 GNN module

(b) Type-2 GNN module

Figure 4: Proposed GNN modules

the graph adjacency powers and thus we study the vector:

$$\boldsymbol{d}^k = diag\left(\boldsymbol{S}^k\right) = \sum_{n=1}^{N} \lambda_n^k |\boldsymbol{u}_n|^2. \tag{14}$$

Since $\boldsymbol{S}$ is the adjacency of the graph, $\mathbf{d}^k$ counts the number of $k-$ length self loops of each node. For instance, when $k = 2$, $\mathbf{d}^k$ indicates the degree of each node, whereas for $k = 3$, it counts the number of triangles each node is involved in, multiplied by a constant factor. For $k = 4$, $\mathbf{d}^k$ holds information about the degrees of $1-$hop and $2-$hop neighbors as well as the $4-$th order cycles. Similar observations are derived by considering larger values of k. Graph adjacency diagonals are not only associated with $k-$hop neighbor degrees but also with motifs that are present in the graph. Overall, $\mathbf{d}^k$ combines $k-$th order degree and subgraph information and the proposed GNN module provides additional knowledge about each node, compared to GNNs with $\mathbf{1}$ inputs. This observation becomes even more valuable, if we consider the significance of subgraph mining in graph theory (Kuramochi & Karypis, 2001; Danisch et al., 2018). Finally, the combined $k-$th order degree and subgraph information, provided by $\mathbf{d}^k$, is associated with the absolute values of the graph adjacency eigenvectors $|\boldsymbol{u}_n|$, whereas degrees alone are connected with $\left(\boldsymbol{u}_n^T\mathbf{1}\right)\boldsymbol{u}_n$.

The following theorem characterizes the expressive power of GNNs with modules as in (11):

Theorem 5.2 *Let $\mathcal{G}$, $\hat{\mathcal{G}}$ be non-isomorphic graphs. If Assumption 2.1 holds, there exists a GNN with diagonal modules as in (11) that tells the two graphs apart.*

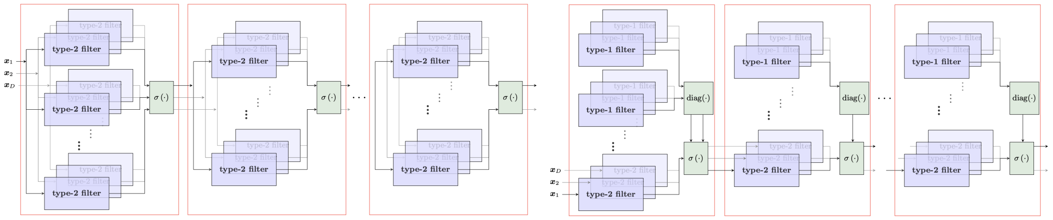
6 Designing powerful GNN architectures

After introducing and analyzing the GNN module in (11), it is time to place it in a broader perspective as part of a GNN architecture. The modules we employ to build the proposed GNN architecture are presented in Fig. 4. Regarding their functionality we provide the following result:

Proposition 6.1 *A GNN designed with the diagonal modules of Fig. 4a in the input layer is equivalent to a standard GNN designed with the modules of Fig. 4b in the input layer, if the input to the modules of Fig. 4b is designed according to:*

$$\boldsymbol{X} = \left[diag\left(\boldsymbol{S}^0\right), diag\left(\boldsymbol{S}^1\right), diag\left(\boldsymbol{S}^2\right), \ldots, diag\left(\boldsymbol{S}^{D-1}\right)\right]. \tag{15}$$

The claim of Proposition 6.1 is fundamental. The diagonal GNN module in (11) is equivalent to a standard GNN module with proper input design. Furthermore, combining propositions 4.1 and 6.1 yields a direct connection between the three considered architectures; GNNs with Gaussian input and variance nonlinearity, GNNs with no input and diagonal operator, and standard GNNs with input as in (15). Guided by these findings we design the GNN architectures presented in Fig. 5. The architecture on the left uses one type of GNN blocks (type-2) and the input is designed by equation (15). Furthermore, it is a symmetric architecture and admits all the favorable properties of symmetric designs. On the other hand, the architecture on the right uses a combination of type-1 and type-2 GNN blocks and designing an input is not necessary. Although the design is not symmetric, it offers reduced number of trainable parameters and reuse of first layer features, which has been observed to benefit convolutional architectures. The expressive power of the proposed architectures is demonstrated in the following theorem:



(a) Type-2 architecture

(b) Type-1 and type-2 architecture

Figure 5: Proposed GNN architectures

Theorem 6.1 *Let $\mathcal{G}$, $\hat{\mathcal{G}}$ be non-isomorphic graphs with graph signals $\boldsymbol{X}$, $\hat{\boldsymbol{X}}$ designed according to* (15). *If Assumption 2.1 holds, then the proposed GNNs in Fig. 5 can tell the two graphs apart.*

Corollary 6.2 *The proposed architectures in Fig. 5 are more expressive compared to GNNs with $\boldsymbol{x} = \boldsymbol{1}$ or $\boldsymbol{x} = \boldsymbol{S}\boldsymbol{1}$ inputs.*

Corollary 6.2 follows from Theorem 6.1 and the fact that both $\text{diag}\left(\boldsymbol{S}^0\right) = \boldsymbol{1},\ \text{diag}\left(\boldsymbol{S}^2\right) = \boldsymbol{S}\boldsymbol{1}$ are included in the proposed input $\boldsymbol{X}$, defined in (15).

7 Simulations

In this section we assess the performance of the proposed GNN architectures in the task of graph classification. In particular, we use graph isomorphism and graph classification datasets and compare against GIN initialized with $\boldsymbol{x} = \boldsymbol{1}$ (Xu et al., 2019), denoted as GIN_1 and GIN modified according to our proposed architectures, i.e., initialized according to equation (15), denoted as GIN_{plus}.

7.1 The CSL dataset

Our first experiment involves the Circular Skip Link (CSL) dataset, which was introduced in (Murphy et al., 2019) to test the expressiveness of GNNs; it is the golden standard when it comes to benchmarking GNNs for isomorphism (Dwivedi et al., 2020). CSL is a symmetric graph dataset. It contains 150 4-regular graphs, where the edges form a cycle and contain skip-links between nodes. A schematic representation of the CSL graphs can be found in Appendix K. Each graph consists of 41 nodes and 164 edges and belongs to one of 10 classes. All the nodes have degree 4 and thus $\boldsymbol{x} = \boldsymbol{1}$ is an eigenvector of every graph and orthogonal to all the remaining eigenvectors. As a result the degree vector is uninformative and so is any message passing operation of the degree.

GNNs initialized with $\boldsymbol{x} = \boldsymbol{1}$ and the WL algorithm fail to provide any essential information for this set of graphs and the classification task is completely random, as shown in Table 4. The proposed GNN architectures, on the other hand, have no issue in dealing with this dataset. In particular a single diagonal GNN module with parameters $(h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8, h_9) = (0, 1, -\frac{1}{2}, \frac{1}{3}, -\frac{1}{4}, \frac{1}{5}, -\frac{1}{6}, \frac{1}{7}, -\frac{1}{8}, \frac{1}{9})$ and $\sigma(\cdot)$ being the linear function, is able to classify these graphs with 100% accuracy. To see this, we present in Table 2 the output $\boldsymbol{1}^T\boldsymbol{y}$ for every class, where $\boldsymbol{y}$ is defined in (11) with the aforementioned parameters. The output is the same for each graph in the same class but different for graphs that belong to different classes. Therefore, perfect classification accuracy is achieved by passing the GNN output to a simple linear classifier or even a linear assignment algorithm.

Table 2: GNN output $\boldsymbol{y}$ for every class of the CSL graphs.

Class									
0	1	2	3	4	5	6	7	8	9
73616	-45968	1059	-30593	-25345	-26001	-17555	-28543	16065	-21163

7.2 Social and biological networks

Next, we test the performance of the proposed architecture with standard social, chemical and bioinformatics graph classification datasets (Errica et al., 2019). The details of each dataset can be found in Table 3. To perform the graph classification task, we train a GNN with 4 layers, each layer consisting of the same number of neurons. The input to each GNN is designed by equation (15) with $K = 10$ and we also pass the $k-$th degree vector. Apart from feeding the output of each layer to the next layer, we also apply a readout function that performs graph pooling. The graph pooling layer generates a global graph embedding from the node representations and passes it to a linear classifier. The nonlinearity is chosen to be the ReLU. An illustration of the used architecture, as well as a detailed description of the experiments, is presented in Appendix K.

To test the performance of the proposed architectures and the baseline we divide each dataset into $50 - 50$ training-testing splits and perform 10-fold cross validation. We measure the micro F1 and macro F1 score for each epoch and present the epoch with the best average result among the 10 folds. The mean and standard deviation of the testing results over 10 shuffles are presented in Table 4.

Table 3: Datasets

Dataset	# Graphs	Average # Vertices	Average # Edges	# Classes	Network Type
CSL	150	41	164	10	Circulant
IMDBBINARY	1,000	20	193	2	Social
IMDBMULTI	1,500	13	132	3	Social
REDDITBINNARY	2000	430	498	2	Social
REDDITMULTI	5000	509	595	5	Social
PTC	344	26	52	3	Bioinformatic
PROTEINS	1,113	39	146	2	Bioinformatic
MUTAG	188	18	20	2	Chemical
NCI1	4110	39	73	2	Chemical

Table 4: Average testing score and standard deviation over 10 shuffles

	Proposed		GIN		GIN$_{\text{plus}}$ (proposed+GIN)	
Dataset	micro F1	macro F1	micro F1	macro F1	micro F1	macro F1
CSL	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$	10 ± 3.3	1.8 ± 0.6	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$
IMDBBINARY	71.7 ± 2.5	71.3 ± 2.7	$\mathbf{74.7 \pm 3.2}$	$\mathbf{74.6 \pm 3.2}$	$\mathbf{71.6 \pm 3.4}$	$\mathbf{71 \pm 3.8}$
IMDBMULTI	46.1 ± 2.8	44.2 ± 3.2	$\mathbf{50.3 \pm 2.8}$	$\mathbf{48 \pm 3.4}$	48.6 ± 2.9	46.1 ± 4.2
REDDITBINARY	87.2 ± 4.1	87.1 ± 4.3	81.6 ± 5.6	81.5 ± 5.7	$\mathbf{89.8 \pm 2.3}$	$\mathbf{89.7 \pm 2.3}$
REDDITMULTI	54 ± 2.2	52.4 ± 2.1	52.4 ± 2.4	50.9 ± 2.4	$\mathbf{55 \pm 1.5}$	$\mathbf{53.6 \pm 1.7}$
PTC	63.6 ± 4.9	61.4 ± 6.9	$\mathbf{65.7 \pm 8.8}$	$\mathbf{65.1 \pm 9.1}$	62.5 ± 5.1	61.4 ± 5.5
PROTEINS	74.2 ± 4.2	73 ± 4	74 ± 4.6	72.3 ± 4.5	$\mathbf{74.3 \pm 4.8}$	$\mathbf{73.1 \pm 4.5}$
MUTAG	89.3 ± 7.3	87.2 ± 9.3	$\mathbf{89.8 \pm 7.6}$	88.6 ± 8.8	$\mathbf{89.8 \pm 8}$	$\mathbf{88.7 \pm 8.6}$
NCI1	74.5 ± 2.1	74.3 ± 2.1	$\mathbf{77.2 \pm 1.9}$	$\mathbf{77.2 \pm 1.9}$	$\mathbf{76.3 \pm 3.7}$	$\mathbf{76.2 \pm 3.8}$

In Table 4 we observe that the proposed architecture and GIN$_{\text{plus}}$ markedly outperform GIN$_1$ in the REDDITBINARY dataset, and also show notable improvement in the REDDITMULTI dataset. GIN$_1$, on the other hand, has a 3% advantage in the IMDBBINARY dataset, whereas in the remaining datasets the performances of the competing algorithms are statistically similar. The latter can be explained, since the vital classification components, of these datasets, are not orthogonal to $\boldsymbol{x} = \mathbf{1}$ and GIN$_1$ is not undergoing critical information loss. Overall, we conclude that properly designed GNNs, as the proposed and GIN$_{\text{plus}}$ can not only demonstrate remarkable performance in graph classification tasks, but can also handle pathological datasets such as the CSL.

8 Conclusion

In this paper we studied the expressive power of GNNs with spectral decomposition tools. We showed that, contrary to common belief, the WL algorithm is not the real limit and proved that GNNs can distinguish between any graphs with at least one different eigenvalue. Furthermore, we explained the limitations of GNNs with all-one inputs and designed GNN architectures that overcome these limitations. Experiments with graph isomorphism and graph classification datasets demonstrated the effectiveness of the proposed architectures. With this work we move one step closer to understanding the properties of GNNs and analyzing their functionality.

REFERENCES

Ralph Abboud, Ismail Ilkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. The surprising power of graph neural networks with random node initialization. In *IJCAI*, 2021.

Waiss Azizian et al. Expressive power of invariant and equivariant graph neural networks. In *International Conference on Learning Representations*, 2020.

Muhammet Balcilar, Pierre Héroux, Benoit Gauzere, Pascal Vasseur, Sébastien Adam, and Paul Honeine. Breaking the limits of message passing graph neural networks. In *International Conference on Machine Learning*, pp. 599–608. PMLR, 2021.

Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. *Advances in neural information processing systems*, 29, 2016.

Dominique Beaini, Saro Passaro, Vincent Létourneau, Will Hamilton, Gabriele Corso, and Pietro Liò. Directional graph networks. In *International Conference on Machine Learning*, pp. 748–758. PMLR, 2021.

Zhengdao Chen, Soledad Villar, Lei Chen, and Joan Bruna. On the equivalence between graph isomorphism testing and function approximation with gnns. *Advances in neural information processing systems*, 32, 2019.

Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems*, 33:13260–13271, 2020.

Miles Cranmer, Peter Melchior, and Brian Nord. Unsupervised resource allocation with graph neural networks. *Proceedings of Machine Learning Research*, 1:1–13, June 2021. URL `http://arxiv.org/abs/2106.09761`.

Maximilien Danisch, Oana Balalau, and Mauro Sozio. Listing k-cliques in sparse real-world graphs. In *Proceedings of the 2018 World Wide Web Conference*, pp. 589–598, 2018.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29:3844–3852, 2016.

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

Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. *arXiv preprint arXiv:1912.09893*, 2019.

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. *Nature Methods*, 17(2):184–192, February 2020.

Fernando Gama, Joan Bruna, and Alejandro Ribeiro. Stability properties of graph neural networks. *IEEE Transactions on Signal Processing*, 68:5680–5695, 2020.

Floris Geerts and Juan L Reutter. Expressiveness and approximation properties of graph neural networks. In *International Conference on Learning Representations*, 2021.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pp. 1263–1272. PMLR, 2017.

Samar Hadou, Charilaos I Kanatsoulis, and Alejandro Ribeiro. Space-time graph neural networks. In *International Conference on Learning Representations*, 2021.

Willem H Haemers and Edward Spence. Enumeration of cospectral graphs. *European Journal of Combinatorics*, 25(2):199–211, 2004.

Ehsan Hajiramezanali, Arman Hasanzadeh, Nick Duffield, Krishna R Narayanan, Mingyuan Zhou, and Xiaoning Qian. Variational graph recurrent neural networks. In *Neural Information Processing Systems (NeurIPS)*, 2019.

William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 1025–1035, 2017.

Dejun Jiang, Zhenxing Wu, Chang Yu Hsieh, Guangyong Chen, Ben Liao, Zhe Wang, Chao Shen, Dongsheng Cao, Jian Wu, and Tingjun Hou. Could graph neural networks learn better molecular representation for drug discovery? a comparison study of descriptor-based and graph-based models. *Journal of Cheminformatics*, 13(1):12, dec 2021.

Nicolas Keriven and Gabriel Peyré. Universal invariant and equivariant graph neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

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

Michihiro Kuramochi and George Karypis. Frequent subgraph discovery. In *Proceedings 2001 IEEE international conference on data mining*, pp. 313–320. IEEE, 2001.

Ron Levie, Wei Huang, Lorenzo Bucci, Michael Bronstein, and Gitta Kutyniok. Transferability of spectral graph convolutional neural networks. *Journal of Machine Learning Research*, 22(272): 1–59, 2021.

Yujia Li, Richard Zemel, Marc Brockschmidt, and Daniel Tarlow. Gated graph sequence neural networks. In *Proceedings of ICLR'16*, 2016.

Vinicius Lima, Mark Eisen, Konstatinos Gatsis, and Alejandro Ribeiro. Resource allocation in large-scale wireless control systems with graph neural networks. *IFAC-PapersOnLine*, 53(2): 2634–2641, 2020.

Xiaorui Liu, Wei Jin, Yao Ma, Yaxin Li, Hua Liu, Yiqi Wang, Ming Yan, and Jiliang Tang. Elastic graph neural networks. In *International Conference on Machine Learning*, pp. 6837–6849. PMLR, 2021.

Andreas Loukas. What graph neural networks cannot learn: depth vs width. In *International Conference on Learning Representations*, 2019.

Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. In *International Conference on Learning Representations*, 2018.

Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. *Advances in neural information processing systems*, 32, 2019a.

Haggai Maron, Ethan Fetaya, Nimrod Segol, and Yaron Lipman. On the universality of invariant networks. In *International conference on machine learning*, pp. 4363–4371. PMLR, 2019b.

Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: higher-order graph neural networks. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, pp. 4602–4609, 2019.

Christopher Morris, Gaurav Rattan, and Petra Mutzel. Weisfeiler and leman go sparse: Towards scalable higher-order graph embeddings. *Advances in Neural Information Processing Systems*, 33: 21824–21840, 2020.

Ryan Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Relational pooling for graph representations. In *International Conference on Machine Learning*, pp. 4663–4673. PMLR, 2019.

Andrei Nicolicioiu, Iulia Duta, and Marius Leordeanu. Recurrent space-time graph neural networks. *Advances in Neural Information Processing Systems*, 32, apr 2019.

Luana Ruiz, Luiz Chamon, and Alejandro Ribeiro. Graphon neural networks and the transferability of graph neural networks. In *Advances in Neural Information Processing Systems*, volume 33, pp. 1702–1712, 2020a.

Luana Ruiz, Fernando Gama, and Alejandro Ribeiro. Gated graph recurrent neural networks. *IEEE Transactions on Signal Processing*, 68:6303–6318, 2020b.

Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Random features strengthen graph neural networks. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pp. 333–341. SIAM, 2021.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. Computational capabilities of graph neural networks. *IEEE Transactions on Neural Networks*, 20 (1):81–102, 2008a.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008b.

Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. Structured sequence modeling with graph convolutional recurrent networks. In *Advances in Neural Information Processing Systems*, pp. 362–373, 2018.

Alexey Strokach, David Becerra, Carles Corbi-Verge, Albert Perez-Riba, and Philip M. Kim. Fast and flexible protein design using deep graph neural networks. *Cell Systems*, 11(4):402–411.e4, October 2020.

Behrooz Tahmasebi, Derek Lim, and Stefanie Jegelka. Counting substructures with higher-order graph neural networks: Possibility and impossibility results. *arXiv preprint arXiv:2012.03174*, 2020.

Petar Veličković, Arantxa Casanova, Pietro Liò, Guillem Cucurull, Adriana Romero, and Yoshua Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, 2018.

Yanbang Wang, Pan Li, Chongyang Bai, and Jure Leskovec. Tedic: Neural modeling of behavioral patterns in dynamic social interaction networks. In *Proceedings of the Web Conference 2021*, WWW '21, pp. 693–705, New York, NY, USA, 2021.

Boris Weisfeiler and Andrei Leman. The reduction of a graph to canonical form and the algebra which appears therein. *NTI, Series*, 2(9):12–16, 1968.

Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. Graph neural networks in recommender systems: a survey. *ACM Computing Surveys (CSUR)*.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=ryGs6iA5Km`.

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 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 10:974–983, June 2018.

A PRELIMINARIES

Networks are naturally represented by graphs $\mathcal{G} := (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, \ldots, N\}$ is the set of vertices (nodes) and $\mathcal{E} = \{(v, u)\}$ are the edges between pairs of nodes. The 1-hop neighborhood $\mathcal{N}(v)$ of node v is the set of nodes $u \in \mathcal{V}$ that satisfy $(u, v) \in \mathcal{E}$. A graph can also be modeled by a Graph Shift Operator (GSO) $\boldsymbol{S} \in \mathbb{R}^{N \times N}$, where $\boldsymbol{S}(i, j)$ quantifies the relation between node i and node j and $N = |\mathcal{V}|$. Popular choices of the GSO is the graph adjacency, the graph Laplacian or weighted versions of them. The nodes of the graph are often associated with graphs signals $\boldsymbol{X} \in \mathbb{R}^{N \times D}$, also known as node attributes, where D is the dimension of each graph signal (feature dimension).

A.1 GRAPH NEURAL NETWORKS (GNNS)

A graph convolution is defined as:

$$\boldsymbol{z} = \sum_{k=0}^{K-1} h_k \boldsymbol{S}^k \boldsymbol{x}, \tag{16}$$

where $\boldsymbol{H}\left(\boldsymbol{S}\right) = \sum_{k=0}^{K-1} h_k \boldsymbol{S}^k$ is a linear filter of length K and $\boldsymbol{x},\ \boldsymbol{z} \in \mathbb{R}^N$ are the input and output of the filter respectively. Let $\boldsymbol{S} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$, be the eigenvalue decomposition of $\boldsymbol{S}$. Then:

$$\boldsymbol{z} = \sum_{k=0}^{K-1} h_k \boldsymbol{U}\boldsymbol{\Lambda}^k \boldsymbol{U}^T \boldsymbol{x} \tag{17}$$

$$\boldsymbol{U}^T\boldsymbol{z} = \sum_{k=0}^{K-1} h_k \boldsymbol{\Lambda}^k \boldsymbol{U}^T \boldsymbol{x} \tag{18}$$

$$\tilde{\boldsymbol{z}} = \sum_{k=0}^{K-1} h_k \boldsymbol{\Lambda}^k \tilde{\boldsymbol{x}}, \tag{19}$$

where $\tilde{\boldsymbol{x}},\ \tilde{\boldsymbol{z}}$ are the frequency representations of $\boldsymbol{x},\ \boldsymbol{z}$ respectively. The frequency representation of the graph filter is $\tilde{\boldsymbol{H}}\left(\boldsymbol{\Lambda}\right) = \sum_{k=0}^{K-1} h_k \boldsymbol{\Lambda}^k$ and can also be written as:

$$\tilde{\boldsymbol{H}}\left(\boldsymbol{\lambda}_i\right) = \sum_{k=0}^{K-1} h_k \boldsymbol{\lambda}_i^k. \tag{20}$$

$\tilde{\boldsymbol{H}}\left(\boldsymbol{\lambda}_i\right)$ is a polynomial on λ_i and $\tilde{\boldsymbol{z}}_i = \tilde{\boldsymbol{H}}\left(\boldsymbol{\lambda}_i\right)\tilde{\boldsymbol{x}}_i$. The simplest form of a Graph Neural Network (GNN) is an array of graph filters followed by point-wise nonlinearities. The l-th layer of the GNN is a graph perceptron, which is described by:

$$\boldsymbol{X}^{(l+1)} = \sigma\left(\sum_{k=0}^{K-1} h_k^{(l)} \boldsymbol{S}^k \boldsymbol{X}^{(l)}\right). \tag{21}$$

Note that here we are using a recursive equation, whereas in the main paper we used $\boldsymbol{X}$ for input and $\boldsymbol{Y}$ for output, to make things simple. Common choices of $\sigma(\cdot)$ are the Rectified Linear Unit (ReLU) activation function, the Leaky ReLU or the hyperbolic tangent function.

A.2 MULTIPLE FEATURE GNNS

As mentioned earlier, the nodes of the graph are usually associated with a graph signal, which is multidimensional, i.e., $D > 1$ and $\boldsymbol{X}^{(l)}$ is a matrix. Although the architecture in (21) can also handle multidimensional graph signals, multiple feature GNNs are commonly used, which are described by the following recursion formula:

$$\boldsymbol{X}^{(l+1)} = \sigma\left(\sum_{k=0}^{K-1} \boldsymbol{S}^k \boldsymbol{X}^{(l)} \boldsymbol{H}_k^{(l)}\right), \tag{22}$$

where $\boldsymbol{H}_k^{(l)} \in \mathbb{R}^{F \times G}$ represents a set of $F \times G$ graph filters. Compared to the architecture in (21), the MIMO GNN employs multiple filters instead of one, and the outputs of the filters are combined to produce a layer output $\boldsymbol{X}^{(l+1)}$ that has feature dimension equal to G.

A.3 Notation

Our notation is summarized in Table 5.

Table 5: Overview of notation.

$\mathcal{G}$	$\triangleq$	Graph
$\mathcal{V}$	$\triangleq$	Set of nodes
$\mathcal{E}$	$\triangleq$	Set of edges
$\boldsymbol{S}$	$\triangleq$	$N \times N$ graph operator
$\boldsymbol{X}$	$\triangleq$	GNN input; represents the $N \times D$ matrix of node attributes (graph signal)
$\boldsymbol{x}$	$\triangleq$	GNN input; represents the vector of node attributes (graph signal)
$\boldsymbol{Z}$	$\triangleq$	matrix output of a linear filter
$\boldsymbol{z}$	$\triangleq$	vector output of a linear filter
$\boldsymbol{Y}$	$\triangleq$	matrix output of a GNN module; $\boldsymbol{Y} = \sigma(\boldsymbol{Z})$
$\boldsymbol{y}$	$\triangleq$	vector output of a GNN module; $\boldsymbol{y} = \sigma(\boldsymbol{z})$
a	$\triangleq$	scalar
$\boldsymbol{a}$	$\triangleq$	vector
$\boldsymbol{A}$	$\triangleq$	matrix
$\boldsymbol{A}^T$	$\triangleq$	transpose of matrix $\boldsymbol{A}$
$\boldsymbol{A}_k$	$\triangleq$	$\boldsymbol{A}[k,:]^T$, k-th row of matrix $\boldsymbol{A}$
$\boldsymbol{a}_k$	$\triangleq$	$\boldsymbol{A}[:,k]$, k-th column of matrix $\boldsymbol{A}$
$\boldsymbol{U}$	$\triangleq$	eigenvector matrix
$\boldsymbol{U}[k,:]$	$\triangleq$	k-th row of $\boldsymbol{U}$ (row vector)
$\boldsymbol{U}[:,k]$	$\triangleq$	k-th column of $\boldsymbol{U}$
$\boldsymbol{u}_k$	$\triangleq$	k-th eigenvector, k-th column of $\boldsymbol{U}$
$\boldsymbol{I}$	$\triangleq$	Identity matrix
$\mathbf{1}$	$\triangleq$	vector of ones
$\mathbf{0}$	$\triangleq$	vector or matrix of zeros
$\lvert \cdot \rvert$	$\triangleq$	point-wise absolute value
$\binom{m}{n}$	$\triangleq$	binomial coefficient

B Relation to other architectures

GNNs have attracted significant attention and numerous architectures have been proposed. The first GNNs by (Scarselli et al., 2008b; Kipf & Welling, 2016; Battaglia et al., 2016; Defferrard et al., 2016) used simple convolutions on static data and graphs, whereas more sophisticated architectures utilize a variety of attention mechanisms (Hamilton et al., 2017; Veličković et al., 2018; Liu et al., 2021). Graph convolutional architectures have also been designed for time varying graphs and signals. Some of them exploit both the graph and time structure (Hajiramezanali et al., 2019; Wang et al., 2021; Hadou et al., 2021), while others employ recurrent architectures (Li et al., 2016; Seo et al., 2018; Nicolicioiu et al., 2019; Ruiz et al., 2020b).

It is often the case that GNNs are presented in literature using different definitions. The GNN by (Kipf & Welling, 2016) for example is written as:

$$\boldsymbol{X}^{(l+1)} = \sigma\left(\boldsymbol{D}^{-1/2}\left(\boldsymbol{S}+\boldsymbol{I}\right)\boldsymbol{D}^{-1/2}\boldsymbol{X}^{(l)}\boldsymbol{H}^{(l)}\right) = \sigma\left(\boldsymbol{D}^{-1/2}\boldsymbol{S}\boldsymbol{D}^{-1/2}\boldsymbol{X}^{(l)}\boldsymbol{H}^{(l)} + \boldsymbol{D}^{-1}\boldsymbol{X}^{(l)}\boldsymbol{H}^{(l)}\right), \tag{23}$$

where $\boldsymbol{S} \in \{0,1\}^{N \times N}$ represents the graph adjacency, $\boldsymbol{D}$ is a diagonal matrix, with $\boldsymbol{D}[i,i]$ being the degree of node i. Matrix $\boldsymbol{D}^{-1/2}\left(\boldsymbol{S}+\boldsymbol{I}\right)\boldsymbol{D}^{-1/2}$ is also a GSO $\boldsymbol{S}'$ and the formula in (23) can be written as:

$$\boldsymbol{X}^{(l+1)} = \sigma\left(\boldsymbol{S}'\boldsymbol{X}^{(l)}\boldsymbol{H}^{(l)}\right), \tag{24}$$

which is a special case of the MIMO GNN in (22), for $K = 2$. Another way that GNNs are represented in literature are via the following equations:

$$\boldsymbol{A}_v^{(l)} = \texttt{AGGREGATE}\left(\left\{\boldsymbol{X}_u^{(l)} : u \in \mathcal{N}(v)\right\}\right) \tag{25}$$

$$\boldsymbol{B}_v^{(l)} = \texttt{COMBINE}\left(\boldsymbol{X}_v^{(l)}, \boldsymbol{A}_v^{(l)}\right) \tag{26}$$

$$\boldsymbol{X}_v^{(l+1)} = \sigma\left(\boldsymbol{H}^{(l)}\boldsymbol{B}_v^{(l)}\right) \tag{27}$$

where $\boldsymbol{X}_v^{(l)}$ is the signal of node v at layer l and the v-th row of the feature matrix $\boldsymbol{X}^{(l)}$, i.e., $\boldsymbol{X}^{(l)} = \begin{bmatrix} \boldsymbol{X}_1^{(l)^T} \\ \vdots \\ \boldsymbol{X}_N^{(l)^T} \end{bmatrix}$. Equivalently, $\boldsymbol{A}_v^{(l)}, \boldsymbol{B}_v^{(l)}$ are rows of matrices $\boldsymbol{A}^{(l)}$, $\boldsymbol{B}^{(l)}$ respectively and represent signals associated with node v. The majority of the architectures based on the equations (25)-(27) can be written as combinations of the GNN modules in (22). Different architectures employ different functions for `AGGREGATE` and `COMBINE`. Popular choices of `AGGREGATE` functions include the mean, the sum, pooling functions or LSTM functions. The `COMBINE` routine, on the other hand, usually utilizes the concatanation or summation function. Let's focus on the case where `AGGREGATE` and `COMBINE` represent the summation or mean function. Then it is easy to see that:

$$\boldsymbol{A}^{(l)} = \boldsymbol{S}\boldsymbol{X}^{(l)} \tag{28}$$

$$\boldsymbol{B}^{(l)} = \boldsymbol{A}^{(l)} + \boldsymbol{X}^{(l)} \tag{29}$$

$$\boldsymbol{X}^{(l+1)} = \sigma\left(\boldsymbol{B}^{(l)}\boldsymbol{H}^{(l)}\right) = \sigma\left(\sum_{k=0}^{1}\boldsymbol{S}^k\boldsymbol{X}^{(l)}\boldsymbol{H}^{(l)}\right), \tag{30}$$

where $\boldsymbol{S}$ is either the graph adjacency or weighted graph adjacency, depending on whether summation or mean function has been used. Therefore the GNN defined in (30) is a special case of the GNN in (22), for $K = 1$ and $\boldsymbol{H}_0^{(l)} = \boldsymbol{H}_1^{(l)}$.

Now consider the GNN defined in (30) that consists of K layers and $\sigma(\cdot)$ is the linear function for the hidden layers and a nonlinear activation function in the output layer, i.e.,

$$\boldsymbol{X}^{(l+1)} = \boldsymbol{S}\boldsymbol{X}^{(l)}\boldsymbol{H}^{(l)} + \boldsymbol{X}^{(l)}\boldsymbol{H}^{(l)} = (\boldsymbol{S} + \boldsymbol{I})\,\boldsymbol{X}^{(l)}\boldsymbol{H}^{(l)}, \text{ for } l = \{0, \ldots, K-2\} \tag{31}$$

$$\boldsymbol{X}^{(l+1)} = \sigma\left(\boldsymbol{S}\boldsymbol{X}^{(l)}\boldsymbol{H}^{(l)} + \boldsymbol{X}^{(l)}\boldsymbol{H}^{(l)}\right), \text{ for } l = K-1 \tag{32}$$

Then it holds that:

$$\boldsymbol{X}^{(l+1)} = (\boldsymbol{S} + \boldsymbol{I})^{l+1}\,\boldsymbol{X}^{(0)}\boldsymbol{H}^{(1)}\cdots\boldsymbol{H}^{(l)}, \text{ for } l = \{0, \ldots, K-2\} \tag{33}$$

$$\boldsymbol{X}^{(l+1)} = \sigma\left(\boldsymbol{S}\boldsymbol{X}^{(l)}\boldsymbol{H}^{(l)} + \boldsymbol{X}^{(l)}\boldsymbol{H}^{(l)}\right), \text{ for } l = K-1 \tag{34}$$

As a result:

$$\boldsymbol{X}^{(K)} = \sigma\left((\boldsymbol{S} + \boldsymbol{I})^K\,\boldsymbol{X}^{(0)}\boldsymbol{H}^{(K-1)}\cdots\boldsymbol{H}^{(0)}\right) = \sigma\left(\sum_{l=0}^{K}\boldsymbol{S}^l\boldsymbol{X}^{(0)}\boldsymbol{H}_l'\right), \tag{35}$$

which again corresponds to the GNN in (22). The last equality holds since

$$(\boldsymbol{X} + \boldsymbol{I})^K = \sum_{l=0}^{K}\binom{K}{l}\boldsymbol{S}^{K-l}, \tag{36}$$

where $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ is the binomial coefficient. Overall there is a direct connection between the GNNs defined by the equations (25)-(27) and the GNNs defined by (22). Furthermore, apropriate selection of GSO and nonlinearities in (22) with respect to the `AGGREGATE` and `COMBINE` functions in (25)-(27) makes the described architectures equivalent.

C Associating the WL algorithm with the spectral decomposition of a graph

In section 3 we observed a connection between the limitations of the WL algorithm and graphs with eigenvectors orthogonal to $\mathbf{1}$. The WL algorithm is initialized with either $\boldsymbol{x} = \mathbf{1}$ or $\boldsymbol{x} = \boldsymbol{S}\mathbf{1}$ and in the remaining iterations this information is propagated (diffused) through the nodes. In particular, at iteration k of the WL algorithm, node i receives a multiset defined as:

$$\mathcal{T}_i^k : \left\{ x_j \in \mathcal{T}_i^k | x_j = \sum_n \lambda_n \left(\boldsymbol{u}_n^T \mathbf{1} \right) \boldsymbol{u}_n(j), \quad j \in \mathcal{N}_i^k \right\}, \tag{37}$$

where $\mathcal{N}_i^k$ denotes the $k-$th neighborhood of node i. If there is one-to-one correspondence between $\mathcal{T}_i^k$ and $\boldsymbol{S}^k\mathbf{1}(i)$ for all nodes i then the WL algorithm can be analyzed by building and comparing the following features for each node:

$$\boldsymbol{X} = \left[\boldsymbol{S}\mathbf{1}, \boldsymbol{S}^2\mathbf{1}, \dots, \boldsymbol{S}^K\mathbf{1} \right] \tag{38}$$

In other words, if the summation operation is a proper hash function for a specific graph, the WL algorithm is equivalent to the feature generation of (38). In that case, we can use the spectral decomposition of $\boldsymbol{S}$ and the analysis of section 3 to characterize the limitations of the WL algorithm. Then the WL algorithm admits the same limitations as the GNNs with $\boldsymbol{x} = \mathbf{1}$ input and it omits the information associated with eigenvectors that are orthogonal to $\mathbf{1}$.

D The stochastic GNN module

In this section we elaborate more on the proposed stochastic GNN module in Fig. 3a. In order to implement it, we can either use the equivalent model in Fig. 3b or we can design an empirical variance model. In practice, the input to the empirical model is a matrix $\boldsymbol{X} \in \mathbb{R}^{N \times M}$ where each element is independently drawn from a Gaussian distribution with zero mean and unit variance and M is the total number of samples. The output of the filter is $\boldsymbol{Z} = \sum_k h_k \boldsymbol{S}^k \boldsymbol{X} \in \mathbb{R}^{N \times M}$ and the maximum likelihood estimate of the empirical covariance of $\boldsymbol{Z}$ takes the form:

$$\boldsymbol{Q} = \frac{1}{M} \boldsymbol{Z}\boldsymbol{Z}^T. \tag{39}$$

Then the GNN output can be written as:

$$\boldsymbol{y} = \text{diag}\left(\boldsymbol{Q}\right) = \frac{1}{M} \text{diag}\left(\boldsymbol{Z}\boldsymbol{Z}^T\right) = \frac{1}{M} \boldsymbol{Z}^2 \mathbf{1} \tag{40}$$

The GNN module of the empirical variance model is illustrated in Fig. 6.

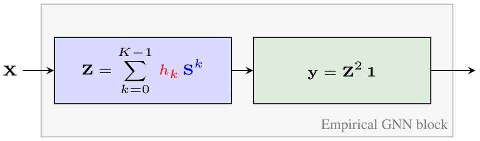

X
$\mathbf{Z} = \sum_{k=0}^{K-1} h_k \mathbf{S}^k$
$\mathbf{y} = \mathbf{Z}^2 \mathbf{1}$
Empirical GNN block

Figure 6: Empirical variance GNN module

D.1 The effect of square nonlinearity

Now we discuss the effect of the square nonlinearity in the representation of the nodes, introduced by the variance operator. As mentioned in section 4 the nonlinearity added by the variance computation allows the proposed GNN to gather information from farther neighborhoods compared to a linear filter or the WL algorithm. To make things more concrete consider the graph in Fig. 7 and let $K - 1 = 2$, which corresponds to running the WL algorithm for 2 iterations and graph filters that process $\boldsymbol{S}$ and $\boldsymbol{S}^2$.

In table 6 we present the representations produced by the stochastic GNN and the WL algorithm for each node of the graph in Fig. 7. In particular we present two iterations of the WL algorithm and the

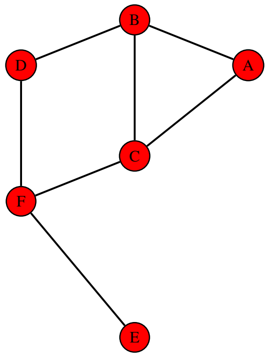


Figure 7: graph

Table 6: GNN vs WL algorithm on the graph in Fig. 7 for $K = 3$.

NODE	WL ALGORITHM	GNN
A	2, 3 3	675
B	3, 2 2 3	1085
C	3, 2 3 3	1134
D	2, 3 3	633
E	1, 3	223
F	3, 1 2 3	896

value that $\boldsymbol{y}$ in (40) converges to, for filter values $(h_0, h_1, h_2) = (3, 5, 7)$. We observe that the WL algorithm represents node A and D with the same value, whereas the output of the stochastic GNN is able to differentiate these two nodes. Overall, the the nonlinearity in the variance operator allows acquiring global information, which can be vital in the resulting node representation of the graph.

D.2 COMPUTING THE COVARIANCE RECURSIVELY

In the main core of the paper we presented 3 almost eguivalent GNN modules; the stochastic module with random Gaussian input, the diagonal module with no input and the standard GNN module with input designed by (15). According to the requirements and constraints of each task, we can employ either of them in a GNN architecture. For instance, in applications where computing the adjacency power diagonals is computationally prohibitive we can use the empirical module in Fig. 6. The drawback is that for systems with high variance, a significant number of samples will be required for the output to converge. This can be mitigated by computing the output in (40) recursively. To be more precise, let $\boldsymbol{z}_m$ be the $m-$th column of filter output $\boldsymbol{Z}$ and $\boldsymbol{Q}_{(M)},\ \boldsymbol{y}_{(M)}$ be the empirical covariance and output after obtaining M samples. Then the recursive equations can be written as:

$$\boldsymbol{Q}_{(M)} = \frac{1}{M}\sum_{m=1}^{M}\boldsymbol{z}_m\boldsymbol{z}_m^T = \frac{1}{M}\sum_{m=1}^{M-1}\boldsymbol{z}_m\boldsymbol{z}_m^T + \frac{1}{M}\boldsymbol{z}_M\boldsymbol{z}_M^T = \frac{M-1}{M}\boldsymbol{Q}_{(M-1)} + \frac{1}{M}\boldsymbol{z}_M\boldsymbol{z}_M^T \tag{41}$$

$$\boldsymbol{y}_{(M)} = \text{diag}\left(\boldsymbol{Q}_{(M)}\right) = \frac{M-1}{M}\text{diag}\left(\boldsymbol{Q}_{(M-1)}\right) + \frac{1}{M}\text{diag}\left(\boldsymbol{z}_M\boldsymbol{z}_M^T\right) = \frac{M-1}{M}\boldsymbol{y}_{(M-1)} + \frac{1}{M}|\boldsymbol{z}_M|^2 \tag{42}$$

Therefore, using $\boldsymbol{y}_{(M)} = \frac{M-1}{M}\boldsymbol{y}_{(M-1)} + \frac{1}{M}|\boldsymbol{z}_M|^2$, allows for online computations and reduces the required memory complexity.

E PROOF OF THEOREM 2.2

To prove Theorem 2.2, consider the GNN module in (2), where $\boldsymbol{H}_k$ is a scalar, i.e.,

$$\boldsymbol{Y} = \sigma\left(\sum_{k=0}^{K-1} h_k \boldsymbol{S}^k \boldsymbol{X}\right) \tag{43}$$

E.1 Case 1: There does not exist permutation matrix $\mathbf{\Pi}$ such that $\boldsymbol{X} = \mathbf{\Pi}\hat{\boldsymbol{X}}$.

Consider an 1− layer GNN with 2 neurons defined by $h_0 = 1,\ h_i = 0,\ i \neq 0$ and $h_0 = -1,\ h_i = 0,\ i \neq 0$, i.e.,

$$\boldsymbol{Y}_1 = \sigma\left(\boldsymbol{X}\right), \quad \boldsymbol{Y}_2 = \sigma\left(-\boldsymbol{X}\right) \tag{44}$$

Summing up the output of the 2 neurons to produce the final GNN output yields $\boldsymbol{Y} = \boldsymbol{Y}_1 + \boldsymbol{Y}_2 = \boldsymbol{X}$ when the $\sigma(\cdot)$ =ReLU$(\cdot)$. As a result, the GNN output is the graph signal and since there does not exist permutation matrix $\mathbf{\Pi}$ such that $\boldsymbol{X} = \mathbf{\Pi}\hat{\boldsymbol{X}}$, this GNN decides that $\mathcal{G}$ and $\hat{\mathcal{G}}$ are different.

E.2 Case 2: Assumption 2.1 holds and $\boldsymbol{VX} \neq \mathbf{0}$.

Let $\mathcal{S} = \{\lambda_1, \ldots, \lambda_N\}$ be the multiset containing the eigenvalues of $\boldsymbol{S}$ and $\hat{\mathcal{S}} = \{\hat{\lambda}_1, \ldots, \hat{\lambda}_N\}$ be the multiset containing the eigenvalues of $\hat{\boldsymbol{S}}$. Note that a multiset can contain multiple instances of a certain element and therefore the eigenvalues of $\boldsymbol{S},\ \hat{\boldsymbol{S}}$ are not required to be distinct. Also let $\{\mu_1, \ldots, \mu_q\}$ be the set of all distinct eigenvalues of $\boldsymbol{S}$ and $\hat{\boldsymbol{S}}$, i.e., $\mu_i \in \mathcal{S} \bigcup \hat{\mathcal{S}}$ and $\mu_i \neq \mu_j,\ \forall\, i \neq j$. Suppose that $\boldsymbol{S},\ \hat{\boldsymbol{S}}$ have at least one different eigenvalue, i.e., there exists μ_k such that $\mu_k \in \mathcal{S}$ but $\mu_k \notin \hat{\mathcal{S}}$.

Recall from Appendix A that a graph filter can be represented in the frequency domain by:

$$\tilde{\boldsymbol{H}}\left(\lambda_i\right) = \sum_{k=0}^{K-1} h_k \lambda_i^k, \tag{45}$$

Then:

$$\begin{bmatrix} \tilde{\boldsymbol{H}}\left(\mu_1\right) \\ \tilde{\boldsymbol{H}}\left(\mu_2\right) \\ \vdots \\ \tilde{\boldsymbol{H}}\left(\mu_q\right) \end{bmatrix} = \begin{bmatrix} 1\ \mu_1\ \mu_1^2 \ldots \mu_1^{K-1} \\ 1\ \mu_2\ \mu_2^2 \ldots \mu_2^{K-1} \\ \vdots \\ 1\ \mu_q\ \mu_q^2 \ldots \mu_q^{K-1} \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ \vdots \\ h_{K-1} \end{bmatrix} = \boldsymbol{Wh} \tag{46}$$

$\boldsymbol{W}$ is a Vandermonde matrix and when $K = q$ the determinant of $\boldsymbol{W}$ takes the form:

$$\det\left(\boldsymbol{W}\right) = \mathbf{\Pi}_{1\leq i<j\leq q}\left(\mu_i - \mu_j\right) \tag{47}$$

Since the values μ_i are distinct, $\boldsymbol{W}$ has full column rank and there exists a graph filter $\boldsymbol{H}\left(\cdot\right)$ with unique parameters $\boldsymbol{h}$ that passes only one eigenvalue (the k-th eigenvalue), i.e.,

$$\tilde{\boldsymbol{H}}\left(\mu_i\right) = \begin{cases} 1, \text{ if } \ i = k \\ 0, \text{ if } \ i \neq k \end{cases} \tag{48}$$

Under this parametrization, the filter $\boldsymbol{H}\left(\cdot\right)$ takes the form $\boldsymbol{H}\left(\boldsymbol{S}\right) = \boldsymbol{VV}^T$, where $\boldsymbol{V}$ is the eigenspace (orthogonal space of the eigenvectors) corresponding to μ_k, and $\boldsymbol{H}\left(\hat{\boldsymbol{S}}\right) = 0$. Then the output of the GNN, for the two graphs, takes the form:

$$\boldsymbol{Y} = \sigma\left(\boldsymbol{H}\left(\boldsymbol{S}\right)\boldsymbol{X}\right) = \sigma\left(\boldsymbol{VV}^T\boldsymbol{X}\right) \tag{49}$$

$$\hat{\boldsymbol{Y}} = \sigma\left(\boldsymbol{H}\left(\hat{\boldsymbol{S}}\right)\hat{\boldsymbol{X}}\right) = \mathbf{0} \tag{50}$$

Under the assumption that $\boldsymbol{V}^T\boldsymbol{X} \neq \mathbf{0}$, it also holds that $\boldsymbol{VV}^T\boldsymbol{X} \neq \mathbf{0}$. As a result $\sigma\left(\boldsymbol{VV}^T\boldsymbol{X}\right) \neq \mathbf{0}$, there does not exist a permutation $\mathbf{\Pi}$ such that $\boldsymbol{Y} = \mathbf{\Pi}\hat{\boldsymbol{Y}}$ and the proposed GNN decides that the two graphs are different. Note $\sigma\left(\boldsymbol{VV}^T\boldsymbol{X}\right) \neq \mathbf{0}$ always holds when for example leaky ReLU is used that allows both positive and negative values to pass. In the case where $\sigma(\cdot)$ =ReLU$(\cdot)$ the proof still holds as long as there is at least one positive value in $\boldsymbol{VV}^T\boldsymbol{X}$. In case $\boldsymbol{VV}^T\boldsymbol{X} \leq \mathbf{0}$ we can without loss of generality consider the filter:

$$\tilde{\boldsymbol{H}}\left(\mu_i\right) = \begin{cases} -1, \text{ if } \ i = k \\ \ \ 0, \text{ if } \ i \neq k \end{cases} \tag{51}$$

that results in $\sigma\left(-\boldsymbol{VV}^T\boldsymbol{X}\right) \neq \mathbf{0}$.

F Proof of Theorems 4.1, 5.2, 6.1:

The proof of Theorems 5.2, 6.1 is equivalent and very similar to the proof of Theorem 4.1. We begin by proving Theorem 5.2. To prove Theorem 5.2 let us consider again the GNN module in (11).

$$\boldsymbol{y} = \sigma\left(\sum_{k=0}^{K-1} h_k \text{diag}\left(\boldsymbol{S}^k\right)\right). \tag{52}$$

If Assumption 2.1 holds, we use the proof of Theorem 2.2 and conclude that there exists a graph filter $\boldsymbol{H}(\cdot)$ with unique parameters $\boldsymbol{h}$ that passes only the k-th eigenvalue, i.e.,

$$\tilde{\boldsymbol{H}}(\mu_i) = \begin{cases} 1, \text{ if } i = k \\ 0, \text{ if } i \neq k \end{cases} \tag{53}$$

Then, $\boldsymbol{H}(\boldsymbol{S}) = \boldsymbol{V}\boldsymbol{V}^T$ and $\boldsymbol{H}\left(\hat{\boldsymbol{S}}\right) = 0$. Then the output $\boldsymbol{y}$ of (52), for the two graphs, takes the form:

$$\boldsymbol{y} = \text{diag}\left(\boldsymbol{H}(\boldsymbol{S})\right) = |\boldsymbol{V}[:,1]|^2 + \cdots + |\boldsymbol{V}[:,m]|^2 = \sum_{i=1}^{m} |\boldsymbol{V}[:,i]|^2 \tag{54}$$

$$\hat{\boldsymbol{y}} = \text{diag}\left(\boldsymbol{H}\left(\hat{\boldsymbol{S}}\right)\right) = 0 \tag{55}$$

where $\boldsymbol{V}[:,i]$ is the $i-$th column of $\boldsymbol{V}$. Since $\boldsymbol{V} \neq \boldsymbol{0}$ by definition, there does not exist a permutation $\boldsymbol{\Pi}$ such that $\boldsymbol{y} = \boldsymbol{\Pi}\hat{\boldsymbol{y}}$ and the proposed GNN can tell the two graphs apart. This concludes the proof for Theorem 5.2. Using Proposition 6.1 we prove equivalence of Theorems 5.2 and 6.1 and therefore the proof is the same.

To prove Theorem 4.1 we need one more extra step. In particular we plug the filter, with parametrization as in (53), in equation (9), for $\boldsymbol{S},\ \hat{\boldsymbol{S}}$, i.e.,

$$\text{cov}\left[\boldsymbol{z}; \boldsymbol{S}\right] = \sum_{k=0}^{K-1} h_k \boldsymbol{S}^k \sum_{m=0}^{K-1} h_m \boldsymbol{S}^m = \boldsymbol{V}\boldsymbol{V}^T\boldsymbol{V}\boldsymbol{V}^T = \boldsymbol{V}\boldsymbol{V}^T \tag{56}$$

$$\text{cov}\left[\boldsymbol{z}; \hat{\boldsymbol{S}}\right] = \sum_{k=0}^{K-1} h_k \hat{\boldsymbol{S}}^k \sum_{m=0}^{K-1} h_m \hat{\boldsymbol{S}}^m = \boldsymbol{0}, \tag{57}$$

where the last equality in (56) holds, since $\boldsymbol{V}$ is orthogonal. Then the output $\boldsymbol{y}$ of (10), for the two graphs, can be written as:

$$\boldsymbol{y} = \text{var}\left[\boldsymbol{z}; \boldsymbol{S}\right] = \text{diag}\left(\text{cov}\left[\boldsymbol{z}; \boldsymbol{S}\right]\right) = |\boldsymbol{V}[:,1]|^2 + \cdots + |\boldsymbol{V}[:,m]|^2 = \sum_{i=1}^{m} |\boldsymbol{V}[:,i]|^2 \tag{58}$$

$$\hat{\boldsymbol{y}} = \text{var}\left[\boldsymbol{z}; \boldsymbol{S}\right] = \text{diag}\left(\text{cov}\left[\boldsymbol{z}; \boldsymbol{S}\right]\right) = 0 \tag{59}$$

Again, since $\boldsymbol{V} \neq \boldsymbol{0}$ by definition, there does not exist a permutation $\boldsymbol{\Pi}$ such that $\boldsymbol{y} = \boldsymbol{\Pi}\hat{\boldsymbol{y}}$ and the proposed stochastic GNN can tell the two graphs apart.

G Proof of Proposition 6.1

The output of type-1 module can be cast as:

$$\boldsymbol{y} = \sigma\left(\sum_{k=0}^{K-1} h_k \text{diag}\left(\boldsymbol{S}^k\right)\right) = \sigma\left(\boldsymbol{X}\boldsymbol{h}\right), \tag{60}$$

when $\boldsymbol{X}$ is designed as in (15) and $\boldsymbol{h} = \begin{bmatrix} h_0 \\ \vdots \\ h_{K-1} \end{bmatrix}$ is the vector of filter parameters. The same output can be produced by the type-2 module when $\boldsymbol{H}_k$ is a vector and $K = 1$. On the other hand, a set of

K type-1 modules in the input layer can produce the $\boldsymbol{X}$ in (15). To see this, consider the following type-1 GNN modules.

$$\boldsymbol{y}_i = \sigma\left(\sum_{k=0}^{K-1} h_k^{(i)} \text{diag}\left(\boldsymbol{S}^k\right)\right), \quad i = 0, \ldots, K-1, \tag{61}$$

where

$$h_k^{(i)} = \begin{cases} 1, \text{ if } \; i = k \\ 0, \text{ if } \; i \neq k \end{cases} \tag{62}$$

Concatenating the outputs $\boldsymbol{y}_i$ into $\boldsymbol{W} = [\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{K-1}]$ results in the $\boldsymbol{X}$ in (15) which we can apply to a type-2 module and produce the same output as a type-2 GNN module with input as in (15). □

H Non-isomorphic graphs with the same set of eigenvalues

In the core of this paper, we discussed the ability of GNNs to distinguish between non-isomorphic graphs that have at least one different eigenvalue. This analysis covers the majority of real graphs, since real graphs almost never share the same eigenvalues. However, there exist interesting cases of graphs with the same set of eigenvalues that GNNs can also tell apart. In this section we study these cases and provide interesting results.

H.1 Graphs with the same distinct eigenvalues

We consider the case where $\boldsymbol{S}, \; \hat{\boldsymbol{S}}$ have distinct eigenvalues which are the same, i.e., $\boldsymbol{\Lambda} = \hat{\boldsymbol{\Lambda}}$. Formally:

Assumption H.1 *$\boldsymbol{S}, \; \hat{\boldsymbol{S}}$ have the same distinct eigenvalues, i.e., $\mathcal{S} \subseteq \hat{\mathcal{S}}$ and $\hat{\mathcal{S}} \subseteq \mathcal{S}$, with $\lambda_i \neq \lambda_j$ for all $i, \; j$.*

Lemma H.2 characterizes non-isomorphic graphs with distinct eigenvalues.

Lemma H.2 *When $\boldsymbol{S}, \; \hat{\boldsymbol{S}}$ have the same distinct eigenvalues, $\mathcal{G}, \; \hat{\mathcal{G}}$ are non-isomorphic if and only if there does not exist permutation matrix $\boldsymbol{\Pi}$ and diagonal ± 1 matrix $\boldsymbol{D}$ such that:*

$$\boldsymbol{U} = \boldsymbol{\Pi}\hat{\boldsymbol{U}}\boldsymbol{D}$$

Proof: Let $\boldsymbol{S} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T, \; \hat{\boldsymbol{S}} = \hat{\boldsymbol{U}}\hat{\boldsymbol{\Lambda}}\hat{\boldsymbol{U}}^T$. Since $\boldsymbol{S}, \; \hat{\boldsymbol{S}}$ have the same distinct eigenvalues it holds that $\boldsymbol{\Lambda} = \hat{\boldsymbol{\Lambda}}$. To prove the 'forward' statement assume that $\mathcal{G}, \; \hat{\mathcal{G}}$ are non-isomorphic, i.e., there does not exist permutation matrix $\boldsymbol{\Pi}$ such that $\boldsymbol{S} = \boldsymbol{\Pi}\hat{\boldsymbol{S}}\boldsymbol{\Pi}^T$. If there exist permutation matrix $\boldsymbol{\Pi}$ and ± 1 diagonal matrix D such that $\boldsymbol{U} = \boldsymbol{\Pi}\hat{\boldsymbol{U}}\boldsymbol{D}$, then:

$$\boldsymbol{S} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T = \boldsymbol{\Pi}\hat{\boldsymbol{U}}\boldsymbol{D}\boldsymbol{\Lambda}\boldsymbol{D}\hat{\boldsymbol{U}}^T\boldsymbol{\Pi}^T = \boldsymbol{\Pi}\hat{\boldsymbol{U}}\hat{\boldsymbol{\Lambda}}\hat{\boldsymbol{U}}^T\boldsymbol{\Pi}^T = \boldsymbol{\Pi}\hat{\boldsymbol{S}}\boldsymbol{\Pi}^T.$$

By contradiction when $\boldsymbol{S}, \; \hat{\boldsymbol{S}}$ have the same distinct eigenvalues, $\mathcal{G}, \; \hat{\mathcal{G}}$ are non-isomorphic if there does not exist permutation matrix $\boldsymbol{\Pi}$ and diagonal ± 1 matrix $\boldsymbol{D}$ such that $\boldsymbol{U} = \boldsymbol{\Pi}\hat{\boldsymbol{U}}\boldsymbol{D}$.

To prove the 'backward' statement assume that there does not exist permutation matrix $\boldsymbol{\Pi}$ and diagonal ± 1 matrix $\boldsymbol{D}$ such that $\boldsymbol{U} = \boldsymbol{\Pi}\hat{\boldsymbol{U}}\boldsymbol{D}$. If $\mathcal{G}, \; \hat{\mathcal{G}}$ are isomorphic, i.e., there exists permutation matrix $\boldsymbol{\Pi}$ such that $\boldsymbol{S} = \boldsymbol{\Pi}\hat{\boldsymbol{S}}\boldsymbol{\Pi}^T$, then:

$$\boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T = \boldsymbol{\Pi}\hat{\boldsymbol{U}}\boldsymbol{\Lambda}\hat{\boldsymbol{U}}^T\boldsymbol{\Pi}^T,$$

which implies that $\boldsymbol{u}_n = \pm\boldsymbol{\Pi}\hat{\boldsymbol{u}}_n$ for all n, where $\boldsymbol{u}_n, \; \hat{\boldsymbol{u}}_n$ refer to the columns of $\boldsymbol{U}, \; \hat{\boldsymbol{U}}$ respectively. As a result, $\boldsymbol{U} = \boldsymbol{\Pi}\hat{\boldsymbol{U}}\boldsymbol{D}$ and by contradiction we prove the 'backward' statement which concludes the proof. □

In a nutshell Lemma H.2 states that in order for $\mathcal{G}, \; \hat{\mathcal{G}}$ to be non-isomorphic, while Assumption H.1 holds, the two graphs need to admit different eigenvectors that correspond to the same eigenvalues. As a side note, we mention that $\boldsymbol{S}, \; \hat{\boldsymbol{S}}$ can still span the same columnspace, under row permutation. However, the power on each eigendirection has to be different for them to be non-isomorphic.

We can now extend the results of Theorem 2.2 to the following:

Theorem H.3 *Let $\mathcal{G},\ \hat{\mathcal{G}}$ be non-isomorphic graphs with graph signals $\boldsymbol{X},\ \hat{\boldsymbol{X}}$. There exists a GNN that tells $\mathcal{G}$ and $\hat{\mathcal{G}}$ apart if:*

1. *There does not exist permutation matrix $\boldsymbol{\Pi}$ such that $\boldsymbol{X} = \boldsymbol{\Pi}\hat{\boldsymbol{X}}$, or*
2. *Assumption 2.1 holds and $\boldsymbol{V}^T\boldsymbol{X} \neq \boldsymbol{0}$, or*
3. *Assumption H.1 holds and $\boldsymbol{X}^T\boldsymbol{u}_n \neq \boldsymbol{0}$ for all eigenvectors $\boldsymbol{u}_n$ or $\hat{\boldsymbol{X}}^T\hat{\boldsymbol{u}}_n \neq \boldsymbol{0}$ for all eigenvectors $\hat{\boldsymbol{u}}_n$.*

Proof: The proof for case 1 and 2 can be found in Appendix E. Case 3 includes Assumption H.1, i.e., both $\boldsymbol{S},\ \hat{\boldsymbol{S}}$ have N distinct eigenvalues, where N is the number of nodes at each graph and also $\boldsymbol{S},\ \hat{\boldsymbol{S}}$ share the same eigenvalues. To prove this last part of Theorem 2.2 we consider an $1-$layer GNN with N neurons. Each neuron consists of a graph filter that isolates one eigenvalue and sets it to one, as in Appendix E. Then, each neuron is described by the following set of equations:

$$\boldsymbol{Y}_n = \sigma\left(\boldsymbol{H}_n\left(\boldsymbol{S}\right)\boldsymbol{X}\right), \quad n = 1, \ldots, N \tag{63}$$

$$\tilde{\boldsymbol{H}}_n\left(\lambda_i\right) = \begin{cases} 1, \text{ if } & i = n \\ 0, \text{ if } & i \neq n \end{cases}, \quad n = 1, \ldots, N \tag{64}$$

For the rest of the proof we will assume that $\sigma(\cdot)$ is the linear function. This is without loss of generality since if we double the number of neurons in the layer and set $\sigma(\cdot)$ =ReLU$(\cdot)$ we can produce the same output as the linear function by using the same trick as in Appendix E.1. In particular, N of the graph filters will follow the equations in (64) and the remaining N filters will follow the same equation with -1 instead, as in (51). Then for each eigenvalue we have a pair of filters, one with $+1$ and one with -1 in the filter equations. Summing up the outputs of these neuron pairs will produce an output that is the same as if $\sigma(\cdot)$ was the linear function.

The output of the GNN for the two graphs takes the form

$$\boldsymbol{Y}_n = \boldsymbol{H}_n\left(\boldsymbol{S}\right)\boldsymbol{X} = \boldsymbol{u}_n\boldsymbol{u}_n^T\boldsymbol{X}, \quad n = 1, \ldots, N \tag{65}$$

$$\hat{\boldsymbol{Y}}_n = \boldsymbol{H}_n\left(\hat{\boldsymbol{S}}\right)\hat{\boldsymbol{X}} = \hat{\boldsymbol{u}}_n\hat{\boldsymbol{u}}_n^T\hat{\boldsymbol{X}}, \quad n = 1, \ldots, N \tag{66}$$

$$\boldsymbol{Y}_n = \boldsymbol{u}_n\left[\boldsymbol{u}_n^T\boldsymbol{x}_1, \ldots, \boldsymbol{u}_n^T\boldsymbol{x}_D\right], \quad n = 1, \ldots, N \tag{67}$$

$$\hat{\boldsymbol{Y}}_n = \hat{\boldsymbol{u}}_n\left[\hat{\boldsymbol{u}}_n^T\hat{\boldsymbol{x}}_1, \ldots, \hat{\boldsymbol{u}}_n^T\hat{\boldsymbol{x}}_D\right], \quad n = 1, \ldots, N \tag{68}$$

Now we assume that $\boldsymbol{X}^T\boldsymbol{u}_n \neq \boldsymbol{0}$ for all eigenvectors $\boldsymbol{u}_n,\ n = 1, \ldots, N$. As a result, there exist at least one column in each $\boldsymbol{Y}_n$ that is not equal to the zero column. We can then collect one non-zero column from each $\boldsymbol{Y}_n$ and form a matrix $\boldsymbol{M}$ as:

$$\boldsymbol{M} = \left[\boldsymbol{u}_1\left(\boldsymbol{u}_1^T\boldsymbol{x}_i\right), \ldots, \boldsymbol{u}_N\left(\boldsymbol{u}_N^T\boldsymbol{x}_j\right)\right] = \left[\boldsymbol{u}_1\alpha_1, \ldots, \boldsymbol{u}_N\alpha_N\right] = \boldsymbol{U}\begin{bmatrix} \alpha_1, 0, \ldots, 0 \\ 0, \alpha_2, \ldots, 0 \\ \vdots \\ 0, 0, \ldots, \alpha_N \end{bmatrix} = \boldsymbol{U}\boldsymbol{A}, \tag{69}$$

where $\boldsymbol{x}_i,\ \boldsymbol{x}_j$ are columns of $\boldsymbol{X}$ such that $\boldsymbol{u}_1^T\boldsymbol{x}_i \neq 0$, $\boldsymbol{u}_N^T\boldsymbol{x}_j \neq 0$, $\boldsymbol{A}$ is a diagonal matrix and $\alpha_n \neq 0$ for all n. If we also collect the corresponding columns for each $\hat{\boldsymbol{Y}}_n$ we can form:

$$\hat{\boldsymbol{M}} = \left[\hat{\boldsymbol{u}}_1\left(\hat{\boldsymbol{u}}_1^T\hat{\boldsymbol{x}}_i\right), \ldots, \widehat{\boldsymbol{u}_N}\left(\hat{\boldsymbol{u}}_N^T\hat{\boldsymbol{x}}_j\right)\right] = \left[\hat{\boldsymbol{u}}_1\hat{\alpha}_1, \ldots, \widehat{\boldsymbol{u}_N}\hat{\alpha}_N\right] = \hat{\boldsymbol{U}}\begin{bmatrix} \hat{\alpha}_1, 0, \ldots, 0 \\ 0, \hat{\alpha}_2, \ldots, 0 \\ \vdots \\ 0, 0, \ldots, \alpha_N \end{bmatrix} = \hat{\boldsymbol{U}}\hat{\boldsymbol{A}}, \tag{70}$$

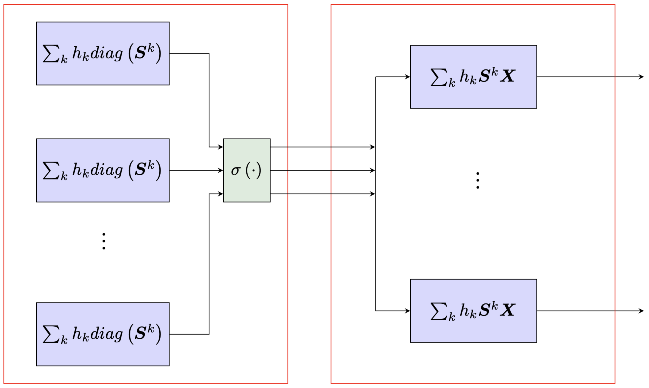

$\sum_k h_k diag\left(\boldsymbol{S}^k\right)$
$\sum_k h_k diag\left(\boldsymbol{S}^k\right)$
$\vdots$
$\sum_k h_k diag\left(\boldsymbol{S}^k\right)$
$\sigma(\cdot)$
$\sum_k h_k \boldsymbol{S}^k \boldsymbol{X}$
$\vdots$
$\sum_k h_k \boldsymbol{S}^k \boldsymbol{X}$

Figure 8: GNN architecture

where $\hat{\boldsymbol{A}}$ is a diagonal matrix but $\hat{\alpha}_n$ are not necessarily non-zero, as $\hat{\boldsymbol{u}}_1^T \hat{\boldsymbol{x}}_i,\ \hat{\boldsymbol{u}}_N^T$ are not necessarily nonzero. Since $\mathcal{G},\ \hat{\mathcal{G}}$ are non-isomorphic and Assumption H.1 holds we can use Lemma H.2. Consequently, there does not exist permutation matrix $\boldsymbol{\Pi}$ and diagonal ± 1 matrix $\boldsymbol{D}$ such that $\boldsymbol{U} = \boldsymbol{\Pi}\hat{\boldsymbol{U}}\boldsymbol{D}$. This implies that there does not exist permutation matrix $\boldsymbol{\Pi}$ such that $\boldsymbol{M} = \boldsymbol{\Pi}\hat{\boldsymbol{M}}$. To complete the proof we consider the output of the considered GNN to be the concatenation of $\boldsymbol{Y}_n,\ n = 1, \ldots, N$. In particular the outputs for $\mathcal{G},\ \hat{\mathcal{G}}$ are:

$$\boldsymbol{Y} = \left[\boldsymbol{Y}_1, \boldsymbol{Y}_2, \ldots, \boldsymbol{Y}_N\right] \tag{71}$$

$$\hat{\boldsymbol{Y}} = \left[\hat{\boldsymbol{Y}}_1, \hat{\boldsymbol{Y}}_2, \ldots, \hat{\boldsymbol{Y}}_N\right]. \tag{72}$$

The columns of $\boldsymbol{M},\ \hat{\boldsymbol{M}}$ are also columns of $\boldsymbol{Y},\ \hat{\boldsymbol{Y}}$. Since there does not exist permutation matrix $\boldsymbol{\Pi}$ such that $\boldsymbol{M} = \boldsymbol{\Pi}\hat{\boldsymbol{M}}$, there does not exist $\boldsymbol{\Pi}$ such that $\boldsymbol{Y} = \boldsymbol{\Pi}\hat{\boldsymbol{Y}}$ and the GNN decides that $\mathcal{G},\ \hat{\mathcal{G}}$ are non-isomorphic. Note that the same analysis is applicable if we assume that $\hat{\boldsymbol{X}}^T \hat{\boldsymbol{u}}_n \neq \boldsymbol{0}$ for all eigenvectors $\hat{\boldsymbol{u}}_n,\ n = 1, \ldots, N$ and therefore it is omitted. Now our proof is complete. □

We also extend the results of Theorems 4.1, 5.2, 6.1 to incorporate the cases where the eigenvalues of the two graphs are the same.

Theorem H.4 *Let $\mathcal{G},\ \hat{\mathcal{G}}$ be non-isomorphic graphs. Then there exists a GNN with modules as in Fig. 3 or as in Fig.4 that tells the two graphs apart, if:*

1. *Assumption 2.1 holds or*
2. *Assumption H.1 holds and*
   (a) *There does not exist permutation matrix $\boldsymbol{\Pi}$ such that $|\boldsymbol{U}| = \boldsymbol{\Pi}|\hat{\boldsymbol{U}}|$, or*
   (b) *$|\boldsymbol{U}^T|\boldsymbol{u}_n \neq \boldsymbol{0}$ for all eigenvectors $\boldsymbol{u}_n$ or $|\hat{\boldsymbol{U}}^T|\hat{\boldsymbol{u}}_n \neq \boldsymbol{0}$ for all eigenvectors $\hat{\boldsymbol{u}}_n$.*

Proof: The proof of case 1 can be found in Appendix F. In order to prove case 2 of Theorem H.4 we use the architecture illustrated in Fig. 8. This GNN is designed with 2 layers, each of them consisting of N neurons. Recall from the previous proofs that there exists a graph filter $\boldsymbol{H}(\boldsymbol{S})$ with unique parameters $\boldsymbol{h}$ that isolates one eigenvalue (the n-th eigenvalue) and sets it to one, i.e.,

$$\tilde{\boldsymbol{H}}(\lambda_i) = \begin{cases} 1, & \text{if } \ i = n \\ 0, & \text{if } \ i \neq k \end{cases} \tag{73}$$

Since the considered graphs have N distinct eigenvalues, we can build the first layer of Fig. 8 with N neurons described by the following set of equations:

$$\boldsymbol{y}_n = \sigma\left(\operatorname{diag}\left(\boldsymbol{H}_n^{(1)}(\boldsymbol{S})\right)\right), \quad n = 1, \ldots, N \tag{74}$$

$$\tilde{\boldsymbol{H}}_n^{(1)}(\lambda_i) = \begin{cases} 1, & \text{if } \ i = n \\ 0, & \text{if } \ i \neq n, \end{cases} \quad n = 1, \ldots, N \tag{75}$$

Under the above parametrization, the filter $\boldsymbol{H}_n^{(1)}(\boldsymbol{S})$ takes the form $\boldsymbol{H}_n^{(1)}(\boldsymbol{S}) = \boldsymbol{u}_n \boldsymbol{u}_n^T$, where $\boldsymbol{u}_n$ is the eigenvector corresponding to the $n-$th eigenvalue of $\boldsymbol{S}$. Then the output of the first layer for the two graphs takes the form:

$$\boldsymbol{y}_n = \sigma\left(\text{diag}\left(\boldsymbol{u}_n \boldsymbol{u}_n^T\right)\right) = |\boldsymbol{u}_n|^2, \quad n = 1, \ldots, N \tag{76a}$$

$$\hat{\boldsymbol{y}}_n = \sigma\left(\text{diag}\left(\hat{\boldsymbol{u}}_n \hat{\boldsymbol{u}}_n^T\right)\right) = |\hat{\boldsymbol{u}}_n|^2, \quad n = 1, \ldots, N \tag{76b}$$

Since both $\boldsymbol{S},\ \hat{\boldsymbol{S}}$ have distinct eigenvalues we can concatenate the output of each neuron and result in layer-1 outputs as:

$$\boldsymbol{Y}^{(1)} = |\boldsymbol{U}|, \quad \hat{\boldsymbol{Y}}^{(1)} = |\hat{\boldsymbol{U}}| \tag{77}$$

If there does not exist permutation matrix $\boldsymbol{\Pi}$ such that $|\boldsymbol{U}| = \boldsymbol{\Pi}|\hat{\boldsymbol{U}}|$, one layer is sufficient and the proposed GNN can tell the two graphs apart.

For the second layer of the GNN in Fig. 8 we consider the following parametrization:

$$\boldsymbol{Y}_n = \left(\boldsymbol{H}_n^{(2)}(\boldsymbol{S})\,\boldsymbol{X}\right), \quad n = 1, \ldots, N \tag{78}$$

$$\tilde{\boldsymbol{H}}_n^{(2)}(\lambda_i) = \begin{cases} 1, \text{ if } \ i = n \\ 0, \text{ if } \ i \neq n' \end{cases} \quad n = 1, \ldots, N, \tag{79}$$

where $\boldsymbol{X} = \boldsymbol{Y}^{(1)} = |\boldsymbol{U}|$ is the output of the first layer. Then the final output of the GNN for the two graphs can be written as:

$$\boldsymbol{Y}_n = \boldsymbol{H}_n^{(2)}(\boldsymbol{S})\,\boldsymbol{X} = \boldsymbol{u}_n \boldsymbol{u}_n^T |\boldsymbol{U}|, \quad n = 1, \ldots, N \tag{80}$$

$$\hat{\boldsymbol{Y}}_n = \boldsymbol{H}_n^{(2)}\left(\hat{\boldsymbol{S}}\right)\hat{\boldsymbol{X}} = \hat{\boldsymbol{u}}_n \hat{\boldsymbol{u}}_n^T |\hat{\boldsymbol{U}}|, \quad n = 1, \ldots, N \tag{81}$$

$$\boldsymbol{Y}_n = \boldsymbol{u}_n \left[\boldsymbol{u}_n^T|\boldsymbol{u}_1|, \ldots, \boldsymbol{u}_n^T|\boldsymbol{u}_N|\right], \quad n = 1, \ldots, N \tag{82}$$

$$\hat{\boldsymbol{Y}}_n = \hat{\boldsymbol{u}}_n \left[\hat{\boldsymbol{u}}_n^T|\hat{\boldsymbol{u}}_1|, \ldots, \hat{\boldsymbol{u}}_n^T|\hat{\boldsymbol{u}}_N|\right], \quad n = 1, \ldots, N \tag{83}$$

If we assume that either $|\boldsymbol{U}^T|\boldsymbol{u}_n \neq \boldsymbol{0}$ for all eigenvectors $\boldsymbol{u}_n$, or $|\hat{\boldsymbol{U}}^T|\hat{\boldsymbol{u}}_n \neq \boldsymbol{0}$ for all eigenvectors $\hat{\boldsymbol{u}}_n$, we can use the same steps as in the proof of Theorem H.3 and show that the proposed GNN decides that the two graphs are different. Note that in layer 1 we can use the stochastic modules in Fig. 3a and the proof still holds, since the filter with parameters as in (75) yields:

$$\text{cov}\left[\boldsymbol{z}; \boldsymbol{S}\right] = \sum_{k=0}^{K-1} h_k \boldsymbol{S}^k \sum_{m=0}^{K-1} h_m \boldsymbol{S}^m = \boldsymbol{u}_n \boldsymbol{u}_n^T \boldsymbol{u}_n \boldsymbol{u}_n^T = \boldsymbol{u}_n \boldsymbol{u}_n^T, \tag{84}$$

and the same output as in (77) can be produced. Also by using Proposition 6.1 we can substitute the modules in the first layer with the modules in Fig. 4b and the proof still holds. □

H.2 Graphs with the same eigenvalues which are not distinct.

The last case appears when the graph adjacencies have the same eigenvalues, but at least one eigenvalue has multiplicity greater than one, in either of the graphs. This case is more complicated, since the two graphs can be non-isomorphic even if there exist permutation matrix $\boldsymbol{\Pi}$ and diagonal matrix $\boldsymbol{D}$ such that $\boldsymbol{U} = \boldsymbol{\Pi}\hat{\boldsymbol{U}}\boldsymbol{D}$ (the condition in Lemma H.2 doesn't hold). Analysis and results for this case are left for future work.

I GNNs and isomorphic graphs

The core of this paper studies the ability of GNNs to distinguish between non-isomorphic graphs. Another important question is whether a GNN can tell if two graphs are isomorphic. The answer is affirmative. GNNs are permutation equivariant architectures and can always detect isomorphic graphs. To make thinks concrete we present the following proposition:

Proposition I.1 . *Let $\mathcal{G}$, $\hat{\mathcal{G}}$ be two isomorphic graphs, i.e.,* $\boldsymbol{S} = \boldsymbol{\Pi}\hat{\boldsymbol{S}}\boldsymbol{\Pi}^T$*. Also let* $\boldsymbol{X}$, $\hat{\boldsymbol{X}}$ *be the graph signals associated with* $\mathcal{G}$, $\hat{\mathcal{G}}$ *that satisfy* $\boldsymbol{X} = \boldsymbol{\Pi}\hat{\boldsymbol{X}}$*. Then any GNN with modules as in* (2) *decides the two graphs are the same.*

Proof: To prove this Proposition, it suffices to show that the output $\boldsymbol{Y}$ in (2) is permutation equivariant. To see this, consider the graph adjacencies $\boldsymbol{S}$ and $\hat{\boldsymbol{S}}$ such that $\hat{\boldsymbol{S}} = \boldsymbol{\Pi}\boldsymbol{S}\boldsymbol{\Pi}^T$, where $\boldsymbol{\Pi}$ is a permutation matrix. Then equation (2) gives:

$$\hat{\boldsymbol{Y}} = \sigma\left(\sum_{k=0}^{K-1} \hat{\boldsymbol{S}}^k \hat{\boldsymbol{X}} \boldsymbol{H}_k\right) \overset{(1)}{=} \sigma\left(\sum_{k=0}^{K-1} h_k \left(\boldsymbol{\Pi}\boldsymbol{S}^k\boldsymbol{\Pi}^T\right)\boldsymbol{\Pi}\boldsymbol{X}\boldsymbol{H}_k\right) \overset{(2)}{=} \sigma\left(\sum_{k=0}^{K-1} h_k \boldsymbol{\Pi}\boldsymbol{S}^k\boldsymbol{X}\boldsymbol{H}_k\right) \quad (85)$$

$$= \sigma\left(\boldsymbol{\Pi}\sum_{k=0}^{K-1} h_k \boldsymbol{S}^k\boldsymbol{X}\boldsymbol{H}_k\right) = \boldsymbol{\Pi}\boldsymbol{Y}, \quad (86)$$

where equality (1) holds because $\left(\boldsymbol{\Pi}\boldsymbol{S}\boldsymbol{\Pi}^T\right)^k = \boldsymbol{\Pi}\boldsymbol{S}^k\boldsymbol{\Pi}^T$ and equality (2) comes from the fact that $\boldsymbol{\Pi}^T\boldsymbol{\Pi} = \boldsymbol{I}$. Therefore $\boldsymbol{Y}$ is permutation equivariant. Overall GNNs with modules as in (2) produce permutation equavariant outputs for isomorphic graphs.

Proposition I.2 . *Let $\mathcal{G}$, $\hat{\mathcal{G}}$ be two isomorphic graphs. Then any GNN with modules as in Fig. 3 or Fig. 4 decides the two graphs are the same.*

Proof: To prove this Proposition, it suffices to show that the output in (11) is permutation equivariant. To see this, consider two graph adjacencies $\boldsymbol{S}$ and $\hat{\boldsymbol{S}}$ such that $\hat{\boldsymbol{S}} = \boldsymbol{\Pi}\boldsymbol{S}\boldsymbol{\Pi}^T$, where $\boldsymbol{\Pi}$ is a permutation matrix. Then equation (11) gives:

$$\hat{\boldsymbol{y}} = \sigma\left(\sum_{k=0}^{K-1} h_k \text{diag}\left(\hat{\boldsymbol{S}}^k\right)\right) \overset{(1)}{=} \sigma\left(\sum_{k=0}^{K-1} h_k \text{diag}\left(\boldsymbol{\Pi}\boldsymbol{S}^k\boldsymbol{\Pi}^T\right)\right) \overset{(2)}{=} \sigma\left(\sum_{k=0}^{K-1} h_k \boldsymbol{\Pi}\text{diag}\left(\boldsymbol{S}^k\right)\right) \quad (87)$$

$$= \sigma\left(\boldsymbol{\Pi}\sum_{k=0}^{K-1} h_k \text{diag}\left(\boldsymbol{S}^k\right)\right) = \boldsymbol{\Pi}\boldsymbol{y}, \quad (88)$$

where equality (1) holds because $\left(\boldsymbol{\Pi}\boldsymbol{S}\boldsymbol{\Pi}^T\right)^k = \boldsymbol{\Pi}\boldsymbol{S}^k\boldsymbol{\Pi}^T$ and equality (2) comes from the fact that $\text{diag}\left(\boldsymbol{\Pi}\boldsymbol{S}\boldsymbol{\Pi}^T\right) = \boldsymbol{\Pi}\text{diag}\left(\boldsymbol{S}\right)$. The output $\boldsymbol{y}$ is permutation equivariant and we can conclude that the proposed architectures produce permutation equivariant outputs for isomorphic graphs.

J GNNS VS SPECTRAL DECOMPOSITION

In this paper we discuss the ability of GNNs to distinguish between different graphs. Our analysis uses spectral decomposition tools and provides conditions under which a GNN can tell two graphs apart. These conditions are related to the eigenvalues and the eigenvectors of the graph operators. Therefore, it is natural to study the similarities and differences of GNNs and spectral decomposition algorithms.

J.1 THE TWO GRAPHS HAVE AT LEAST ONE DIFFERENT EIGENVALUE

As explained in the main part of the paper there always exist a GNN that can distinguish between a pair of graphs with at least one different eigenvalue. Furthermore, computing the eigenvalues of the two graphs can also attest that the two graphs are non-isomorphic. Therefore the two approaches are equally powerful. The difference lies in the fact that a GNN needs to be trained to perform the isomorphism test, whereas the spectral decomposition is unsupervised. On the other hand computing the spectral decomposition for real graphs can be computationally very challenging.

J.2 THE TWO GRAPHS HAVE THE SAME SET OF EIGENVALUES THAT ARE DISTINCT

This case is a bit more complicated. Since the eigenvalues are the same one must resort in the eigenvectors to distinguish between the graphs. When the eigenvalues are distinct, the eigenvectors of the graph are unique up to a sign for each eigenvector. To be more precise, let $\mathcal{G},\ \hat{\mathcal{G}}$ be isomorphic graphs with eigenvectors $\boldsymbol{U},\ \hat{\boldsymbol{U}}$ respectively. Then it holds that:

$$\boldsymbol{U} = \boldsymbol{\Pi}\hat{\boldsymbol{U}}\boldsymbol{D}, \tag{89}$$

where $\boldsymbol{\Pi}$ is a permutation matrix and $\boldsymbol{D}$ is a diagonal matrix with elements ± 1. We observe the following:

Remark J.1 *When Assumption H.1 holds, the eigenvectors of isomorphic graphs are not permutation equivariant, since there exists a sign ambiguity for each eigenvector. On the contrary, the produced GNN node embeddings are always permutation equivariant, according to Propositions I.2 and I.1. In other words GNNs always produce equivariant node embeddings for isomorphic graphs, which is not the case for the spectral decomposition.*

If $\mathcal{G},\ \hat{\mathcal{G}}$ are non-isomorphic the story is different. According to Lemma H.2, there does not exist permutation matrix $\boldsymbol{\Pi}$ such that $\boldsymbol{U} = \boldsymbol{\Pi}\hat{\boldsymbol{U}}\boldsymbol{D}$ and the GNNs detect non-isomorphic graphs under Theorem H.3, or Theorem H.4. Let us focus on the conditions of Theorem H.4 i.e.,

(a) There does not exist permutation matrix $\boldsymbol{\Pi}$ such that $|\boldsymbol{U}| = \boldsymbol{\Pi}|\hat{\boldsymbol{U}}|$,

(b) $|\boldsymbol{U}^T|\boldsymbol{u}_n \neq \boldsymbol{0}$ for all eigenvectors $\boldsymbol{u}_n$ or $|\hat{\boldsymbol{U}}^T|\hat{\boldsymbol{u}}_n \neq \boldsymbol{0}$ for all eigenvectors $\hat{\boldsymbol{u}}_n$.

We see that these conditions involve the eigenvectors of the graphs and therefore we can construct an eigen-based algorithm with the same guarantees. Note that these guarantees are only sufficient and there might be cases where the GNNs can distinguish between non-isomorphic graphs, whereas an algorithm based on the above conditions might fail. Furthermore, calculating the complete set of eigenvectors of a real graph might be computationally prohibitive.

J.3 The two graphs have the same set of eigenvalues that are not distinct

The GNN analysis for this case is relegated for future work. Regarding the spectral decomposition, we distinguish between two scenarios:

Scenario 1: The eigenvalues of the two graphs have the same multiplicities. In that scenario we need to resort to eigenvectors, which are not unique. Therefore detecting non-isomorphic graphs is challenging.

Scenario 1: The eigenvalues of the two graphs have different multiplicities. The graphs of this scenario are by definition non-isomorphic and can be detected by an eigenvalue check. On the other hand, it is not clear whether a GNN can distinguish between graphs with these spectral properties.

J.4 Stability and discriminability of GNNs

From our discussion so far, we noticed the similarities and differences between the functionality of GNNs and the spectral decomposition of the graph. There is one more fundamental difference that has not yet been discussed and involves the stability and discriminability properties of GNNs (Gama et al., 2020). In particular, a GNN is stable under small perturbations of the graph operator, i.e., the output of a GNN is similar for 'similar' graphs. On the other hand, small perturbations of the graph can result in essential changes in the eigenvalues and eigenvectors of the graph operator, which makes the spectral decomposition more unstable. Therefore, there seems to be a stability vs discriminability trade-off between GNNs and spectral decomposition. However, the architectural nonlinearities allow GNNs to be both stable and discriminative.

To recap, the conditions of this paper involve the eigenvalues and eigenvectors of the graph operator. Compared to eigen-based algorithms there is a clear advantage of GNNs when the eigenvalues are exactly the same with the same multiplicities. This is due to the fact that the eigenvectors of a graph operator are not unique and therefore isomorphic graphs do not admit permutation equivariant eigenvectors, whereas GNNs always produce permutation equivariant node embeddings for isomorphic graphs. On the other hand, when the eigenvalues are different, GNNs and the spectral decomposition are equally powerful, when there is at least one different eigenvalue in the graph, whereas the spectral decomposition has an advantage when the eigenvalues are the same but with different multiplicities. Furthermore, GNNs are robust to small changes of the graph, which is not the case for the spectral decomposition. Finally, the spectral decomposition is computationally heavy and unsupervised, but GNNs are lighter to execute and require training.

K Experimental details

In this appendix we provide further details on the experiments presented in the main paper.

K.1 Experiments associated with the graphs in Figs. 1 and 2

In Tables 7, 8 we present the eigenvalues and the sum of the corresponding eigenvectors of the graphs in Figs. 1, 2 respectively.

Table 7: Eigenvalue and eigenvector information for the graphs in Fig. 1.

Graph		n					
		1	2	3	4	5	6
$\mathcal{G}$	λ_n	3	1	-2	-2	0	0
	$\boldsymbol{u}_n^T\mathbf{1}$	-2.45	0	0	0	0	0
$\hat{\mathcal{G}}$	$\hat{\lambda}_n$	3	-3	0	0	0	0
	$\hat{\boldsymbol{u}}_n\mathbf{1}$	-2.45	0	0	0	0	0

Table 8: Eigenvalue and eigenvector information for the graphs in Fig. 2.

Graph		n									
		1	2	3	4	5	6	7	8	9	10
$\mathcal{G}$	λ_n	2.303	1.618	1.303	1	0.618	-2.303	-1.618	-0.618	-1	-1.303
	$\boldsymbol{u}_n^T\mathbf{1}$	3.048	0	0	-0.816	0	0	0	0	0	-0.210
$\hat{\mathcal{G}}$	$\hat{\lambda}_n$	2.303	1.861	1	0.618	0.618	0.254	-1.303	-1.618	-1.618	-2.115
	$\hat{\boldsymbol{u}}_n\mathbf{1}$	3.048	0	-0.816	0	0	0	-0.210		0	0

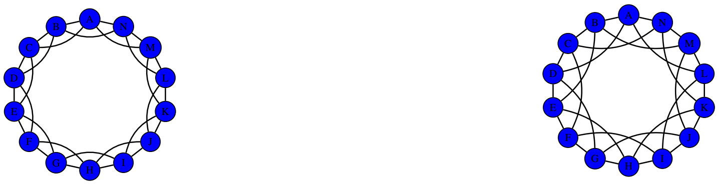
We observe that $\mathcal{G}$ and $\hat{\mathcal{G}}$ in both figures admit a different set of eigenvalues. However, the eigenvectors that correspond to the eigenvalues that differentiate them are orthogonal to the vector of all-ones (they sum up to zero). Therefore the WL algorithm and GNNs with $\boldsymbol{x} = \mathbf{1}$ input fail to tell them apart.

K.2 Details on the experiments of section 7

In Fig. 9 we present a paradigm of two graphs in the CSL dataset that belong to different classes. It is clear from the figure that the two graphs consist of nodes that all have degree equal to 4. Therefore $\boldsymbol{x} = \mathbf{1}$ is an eigenvector of both graphs and orthogonal to the remaining eigenvectors. Any valuable information that tells the two graphs apart is lost when we run the WL algorithm or feed a GNN with $\boldsymbol{x} = \mathbf{1}$.

Next, we present the details on the experiments of section 7.2. For the most part we use the specifications suggested in (Xu et al., 2019). In particular we train a 4-layer graph neural network where the output of each layer and the input are passed through a graph pooling layer and then a linear classifier. A schematic representation of the considered architecture is presented in Fig. 10. The nonlinearity used in our experiments is the ReLU and the readout function that performs graph pooling is $\mathbf{1}^T\boldsymbol{X}^{(l)}$ for $l = 0, \ldots, 5$. $\boldsymbol{X}^{(l)}$ represents the output of layer l with $\boldsymbol{X}^{(0)} = \boldsymbol{X}$. For each type-2 GNN block we only use 1 tap for $k = 1$. This is due to the fact that we pass the output of every layer to the final classifier so additional taps might be redundant.

To train the proposed architecture we use Adam optimizer with learning rate equal to 10^{-2}, batch size equal to 128 and dropout ratio equal to 0.5. The training is performed over 200 epochs with 50



(a) $\mathcal{G}$

(b) $\hat{\mathcal{G}}$

Figure 9: CSL graphs

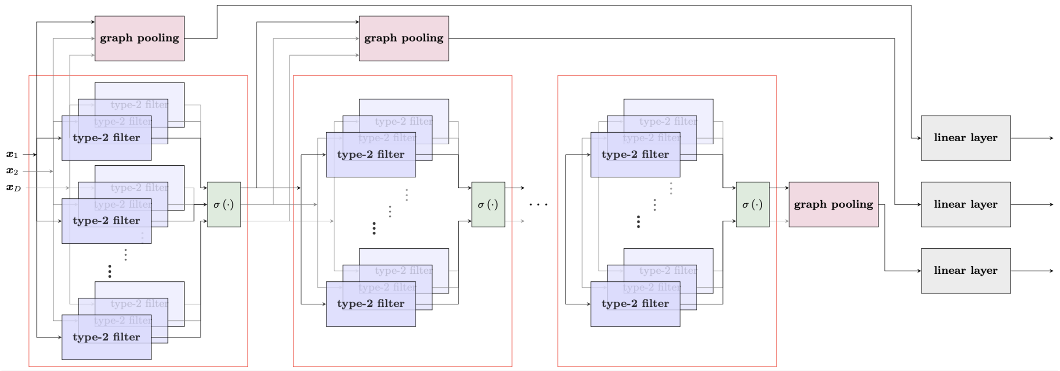


Figure 10: GNN architecture

iterations per epoch. To assess the performance of the proposed architecture we divide each dataset into $50-50$ training-testing splits and apply 10-fold cross validation. The only parameter we tune is the hidden dimension for each layer. In particular, the number of modules for each layer is the same and we tune over $\{8, 16, 32, 64, 128, 256\}$ modules.

We also compare our proposed architecture with GIN (Xu et al., 2019) initialized with $\boldsymbol{x} = \mathbf{1}$ and GIN initialized according to equation (15). We use the publicly available code[1] provided by the authors. We use the exact same specification for fair comparisons and tune the hidden layer over $\{8, 16, 32, 64, 128, 256\}$ dimensions.

All experiments are conducted in a linux server with NVIDIA RTX 3080 GPU. The data[12] are publicly available and code of the proposed architectures with all the experiments can be found in this repository[3].

[1] https://github.com/weihua916/powerful-gnns

[2] https://pytorch-geometric.readthedocs.io/en/latest/

[3] https://github.com/tempcode100/gnns-are-powerful