Journal of Machine Learning Research 26 (2025) 1-35 Submitted 2/23; Revised 9/24; Published 1/25

Orthogonal Bases for Equivariant Graph Learning with Provable k-WL Expressive Power

Jia He JHE58@HAWK.IIT.EDU
Department of Applied Mathematics
Illinois Institute of Technology
Chicago, IL 60616-3793, USA

Maggie X. Cheng* MAGGIE.CHENG@IIT.EDU
Department of Applied Mathematics
Illinois Institute of Technology
Chicago, IL 60616-3793, USA

Editor: Joan Bruna

Abstract

Graph neural network (GNN) models have been widely used for learning graph-structured data. Due to the permutation-invariant requirement of graph learning tasks, a basic element in graph neural networks is the invariant and equivariant linear layers. Previous work (Maron et al., 2019b) provided a maximal collection of invariant and equivariant linear layers and a simple deep neural network model, called k-IGN, for graph data defined on k-tuples of nodes. It is shown that the expressive power of k-IGN is at least as good as the k-Weisfeiler-Leman (WL) algorithm in graph isomorphism tests. However, the dimension of the invariant layer and equivariant layer is the k-th and $2k$-th bell numbers, respectively. Such high complexity makes it computationally infeasible for k-IGNs with $k \geq 3$.

In this paper, we show that a much smaller dimension for the linear layers is sufficient to achieve the same expressive power. We provide two sets of orthogonal bases for the linear layers, each with only $3(2^k - 1) - k$ basis elements. Based on these linear layers, we develop neural network models GNN-a and GNN-b and show that for the graph data defined on k-tuples of data, GNN-a and GNN-b achieve the expressive power of the k-WL algorithm and the $(k+1)$-WL algorithm in graph isomorphism tests, respectively. In molecular prediction tasks on benchmark datasets, we demonstrate that low-order neural network models consisting of the proposed linear layers achieve better performance than other neural network models. In particular, order-2 GNN-b and order-3 GNN-a both have 3-WL expressive power, but use a much smaller basis and hence much less computation time than known neural network models.

Keywords: Permutation invariant/equivariant, Expressive power, Graph isomorphism, Weisfeiler-Leman, Graph neural networks

1. Introduction

Graph data arise from many different fields of science and engineering, such as the study of social networks, citation networks, and molecular chemistry. Learning from graph-structured data has become the core technology to enable powerful graph data analysis.

*. Corresponding author

Among others, graph neural networks have received the most attention due to their superior performance in graph learning tasks (e.g., Gilmer et al., 2017; Xu et al., 2019; Morris et al., 2019), and some early successes have been reported in molecular prediction and social network applications (Maron et al., 2019a).

Recent empirical success of graph neural networks also brought interest to theoretical studies. Two lines of research attracted the most interest: one line of work focuses on the universal approximation of permutation-invariant functions by GNNs, and the other line focuses on the assessment of the expressive power of GNNs, mainly via graph isomorphism tests. Since the Weisfeiler-Leman algorithm has been widely used in graph isomorphism tests (Weisfeiler and Leman, 1968; Weisfeiler, 1976), a common practice is to measure the expressive power of neural networks in equivalent k-WL expressiveness. Chen et al. (2019b) introduced a language of sigma-algebra and unified the two lines by showing the equivalence between graph isomorphism testing and function approximation. For graphs represented as high-order tensors, Azizian and Lelarge (2021) characterized the set of functions that can be approximated by finite dimension Linear GNNs and Folklore GNNs at fixed order of tensors.

The search for neural network models that are highly expressive and able to approximate permutation invariant functions on graphs has become a recent quest. There are several independent lines of work towards this objective. Morris et al. (2019) developed neural network models to generalize message passing to higher orders so that they possess the expressiveness of higher-order WL tests. Although it surpassed the 1-WL expressiveness of the original message passing network (Gilmer et al., 2017), it is still computationally prohibitive to implement models with 3-WL expressiveness.

In a recent study, Maron et al. (2019b) introduced a hierarchy of Invariant Graph Networks (IGNs) that leverage higher-order tensors. An IGN employing order-k tensors is referred to as k-IGN. These networks typically use a maximal set of orthogonal bases to construct linear layers, ensuring all equivalence relations defined on the multiset index are covered.

The exhaustiveness comes at the cost of large dimensions for the linear layers. Although k-IGNs achieved k-WL expressiveness, it is impractical for networks with $k \geq 3$ in practice. Maron et al. (2019a) proposed a simple GNN model using matrix multiplication and it achieved 3-WL expressiveness. It is not easy to have a model with higher than 3-WL expressiveness due to the complexity. In summary, due to the limited expressive power of low-order GNNs, and the high computational complexity of high-order GNNs, there is still large room to improve the expressive power of GNN models and reduce the model complexity.

In this work, we propose a new set of orthogonal bases to construct linear layers for the graph neural networks. This set of orthogonal bases is not exhaustive in terms of including all equivalence relations defined on the multiset index,

but is enough to provide sufficient discriminative power for the neural networks. The discriminative power of graph neural network models can be assessed based on their k-WL expressiveness in graph isomorphism testing. The new orthogonal bases have much smaller dimensions than the maximal set and yet have the same expressive power in graph isomorphism testing. Specifically, we propose two types of GNN models: our type-a GNN models of order k have k-WL expressiveness, and type-b GNN models of order k have k-

FWL expressiveness. To our knowledge, this is the smallest set of bases for achieving the same k-WL expressive power, which further leads to the following comparison with previous models:

1. It is known that k-Folklore-WL (k-FWL) is equivalent to $(k+1)$-WL for $k \geq 2$ (Morris et al., 2019; Grohe, 2017; Cai et al., 1992). The general k-IGN is shown to have k-WL expressiveness for $k \geq 2$ (Maron et al., 2019b), and the PPGN based on matrix multiplication is shown to have 3-WL expressiveness (Maron et al., 2019a). Therefore, our order-k GNN-a has the same expressiveness as k-IGN, and order-k GNN-b has the same expressiveness as $(k+1)$-IGN. Most importantly, the complexity of the proposed GNN models is much less than IGNs of the same order. Our order-2 GNN-b also has 3-WL power but is much less complex than PPGN (Maron et al., 2019a).

2. It is shown that the popular message passing networks (Gilmer et al., 2017) cannot distinguish between graphs that are indistinguishable by the 2-dimensional Weisfeiler-Leman algorithm (Morris et al., 2019; Xu et al., 2019). Since the message-passing network has only 2-WL power, our order-2 GNN-b and order-3 GNN-a models are both more powerful than the family of message-passing networks.

2. Related work

Deep neural networks for graph learning The pioneering work for permutation invariant networks started from characterizing permutation equivariant and invariant linear layers for sets (see Zaheer et al., 2017). About the same time frame, PointNet (Charles et al., 2017) was proposed to achieve permutation invariance on point sets. Hartford et al. (2018) extended it to learning interactions across sets, which can be conveniently modeled as a graph. It uses a simple basis to produce permutation equivariant layers. Because of the use of the basis, the trainable parameters are independent of graph size. However, due to the extreme simplicity, the expressive power is very limited although the network is permutation invariant. Maron et al. (2019b) provided a full characterization of all permutation invariant and equivariant linear layers. The linear layers use a maximal set of bases and can work with high-order tensors. Maron et al. (2019a) built upon the k-IGN from Maron et al. (2019b) and present a reduced 2-order network containing a matrix multiplication operation and proved it has 3-WL expressive power.

Another popular family of networks is the message-passing neural networks. Gilmer et al. (2017) provides a common framework called *MPNN* that fits many existing GNN models, including Duvenaud et al. (2015), Kipf and Welling (2017), Li et al. (2016), Battaglia et al. (2016), and Schütt et al. (2017). The features at each node are updated iteratively by differentiable functions that take as input the previous features of the node itself, its neighborhood, and edge features (if they exist). Recent advances (Morris et al., 2019; Bodnar et al., 2021b) made progress from the 1-WL expressiveness, but it is still computationally prohibitive for models with higher orders.

Although graph neural networks are the focus of recent studies and there are many noteworthy works (e.g., Scarselli et al., 2009; Niepert et al., 2016; Kondor et al., 2018; Vignac et al., 2020; Papp et al., 2021; Sun et al., 2019; Mallea et al., 2019; Veličković et al., 2018;

Gao and Ji, 2019; Li et al., 2020; Zhang and Li, 2021; Chen et al., 2020; Liu et al., 2020; Bouritsas et al., 2023), they are not the only representative in graph learning. Kernel-based methods (e.g., Kondor et al., 2009; Shervashidze et al., 2011; Yanardag and Vishwanathan, 2015; Neumann et al., 2016) offered a competitive alternative. Yanardag and Vishwanathan (2015), for example, provided a unified framework to learn latent representations of substructures by using techniques from language modeling. Kernel-based methods are effective in learning *specific* substructures. However, since the substructure space is of dimension exponential to the number of nodes, it is infeasible to use kernel-based methods to learn a large graph.

Expressive power of neural networks To study the expressive power of neural networks, one line of research examines the ability of graph neural networks to approximate permutation-invariant functions on graphs (Keriven and Peyré, 2019; Maron et al., 2019c), and another line of research studies their ability to distinguish non-isomorphic graphs (Grohe, 2017; Huang and Villar, 2021; Xu et al., 2019; Morris et al., 2019; Maron et al., 2019a). Morris et al. (2020) proposed local variants of the k-dimensional Weisfeiler-Leman algorithm by considering only a subset of the original neighborhood, making them more scalable and less prone to overfitting. One variant is shown to be more powerful than the standard k-WL while taking the underlying graph's sparsity into account. Recent work Chen et al. (2019b) proposed to study the expressive power of graph neural networks by their ability to count graph substructures, and it shows the equivalence between graph isomorphism testing and approximation of permutation-invariant functions by GNNs. Azizian and Lelarge (2021) gave a general theoretical framework to compare the expressive power of the GNN architectures. In particular, it shows that tensor-based GNNs augmented with matrix multiplication, which are called Folklore Graph Neural Networks (FGNN) in the paper, are the most expressive architectures proposed so far. Our type-b GNNs achieve the same expressive power for a given tensor order while substituting matrix multiplication with faster Boolean functions.

Recent advances The study of invariant graph networks has been extended to the spectral domain by leveraging invariant features derived from eigenvectors and eigenvalues. Zhang et al. (2024) introduced Spectral Invariant Graph Network and showed that spectral k-IGN is as expressive as k-WL, and spectral k-FGNN is as expressive as k-FWL for $k \geq 2$.

On the expressive power of graph neural networks, recent work has also considered geometric graphs embedded in Euclidean space. Such graphs often occur in biochemical or physical systems and are characterized by both geometry and relational structure. A geometric version of the WL test named GWL test for discriminating geometric graphs has been proposed in Joshi et al. (2023), in which the invariant or equivariant properties are defined in terms of physical symmetries rather than node permutation. A similar conclusion exists for geometric graphs, that is, high-order tensors enable maximally powerful geometric GNNs.

3. Linear layers

The proposed graph neural network model falls under the category of order-k Linear GNN as classified by Azizian and Lelarge (2021). In this model, graphs are represented as tensors

of order-k and the network consists of a series of linear layers, i.e.,

$$F = L_T \circ L_{T-1} \dots L_1.$$

Typically, each linear layer is followed by a non-linear activation function. An example of this type of architecture is the k-IGN, which was introduced by Maron et al. (2019b). In the k-IGN model, each layer is represented as a linear function $L : \mathbb{R}^{n^k} \to \mathbb{R}^{n^l}$. Our network follows a similar architecture but differs from k-IGN in terms of the basis functions used.

In this following, we formally introduce our linear basis, the functionality of each basis function, and the initial representation of a graph with tensors. At the end of this section, we give a detailed comparison with previous work Maron et al. (2019b).

3.1 Orthogonal bases for the linear layers

A linear layer $L : \mathbb{R}^{n^k} \to \mathbb{R}^{n^l}$ is a linear combination of a set of linear basis functions plus the bias term,

$$L(X) = \sum_{\gamma} w_\gamma h_\gamma(X) + \sum_{\beta} b_\beta C^\beta, \tag{1}$$

where w_γ and b_β are learnable parameters. The input X is an order-k tensor and the output $L(X)$ is an order-l tensor, with $l \in \{0, 1, \dots, k\}$. $h_\gamma(X) = B^\gamma X$ is a linear basis function. The tensor B^γ has a dimension of n^{k+l}, while the tensor C^β has a dimension of n^l.

B^γ and C^β both have binary entries. An entry in tensor B^γ is indexed by a $(k+l)$-tuple $\mathcal{I} = (i_1, \dots, i_{l+k})$. γ represents an equivalence class of the $(k+l)$-tuple indices. We defer the definition of the equivalence classes to Section 3.2.

For each equivalence class γ, entries in tensor B^γ is set as follows

$$(B^\gamma)_{\mathcal{I}} = \begin{cases} 1, & \text{if } \mathcal{I} \in \gamma, \\ 0, & \text{otherwise.} \end{cases}$$

Similarly, we can set the binary entries in tensor C^β. An entry in tensor C^β is indexed by an l-tuple $\mathcal{J} = (i_1, \dots, i_l)$. β represents an equivalence class of the l-tuple indices. For each equivalence class β, entries in tensor C^β is set as follows

$$(C^\beta)_{\mathcal{J}} = \begin{cases} 1, & \text{if } \mathcal{J} \in \beta, \\ 0, & \text{otherwise.} \end{cases}$$

The complexity of the neural network model depends on the dimension of the function space of $L(X)$, which depends on the cardinality of the set $\{\gamma\}$ and $\{\beta\}$. We use $\dim(\gamma)$ and $\dim(\beta)$ to denote them, respectively.

To generalize to d input features and d' output features, the dimension of $L : \mathbb{R}^{n^k \times d} \to \mathbb{R}^{n^l \times d'}$ is $dd'\dim(\gamma) + d'\dim(\beta)$. The parameter matrices are of dimension $dd'\dim(\gamma)$ and $d'\dim(\beta)$, respectively.

3.2 Equivalence classes

Consider $L : \mathbb{R}^{n^k} \to \mathbb{R}^{n^l}$. A linear basis function $h_\gamma(X) = B^\gamma X$ has a one-to-one correspondence with an equivalence class γ, so the size of the basis is the number of equivalence classes.

An equivalence class is uniquely represented by an equivalence relation on the multiset $\{i_1, \ldots, i_{k+l}\}$. An equivalence relation on the multiset specifies an equality pattern of the elements, which defines a partition of the set, i.e., $i_u = i_v$ for $u, v \in [k+l]$, and $u \neq v$ if and only if i_u, i_v belong to the same subset in the partition. Since all the $(k+l)$-tuple indices that satisfy the same equivalence relation belong to the same equivalence class, we use the equivalence relation, which is given in the form of set-partition, to represent an equivalence class.

The equivalence class γ decides the functionality of the basis function $h_\gamma(X) = B^\gamma X$. The functionality of the basis function can be understood from a matrix-vector multiplication point of view. Recall that the B^γ in (1) has dimension n^{k+l}. The input order-k tensor can be viewed as a flattened vector of n^k, similarly, the output tensor can be viewed as a flattened vector of n^l, and the tensor B^γ can be viewed as a matrix of $n^l \times n^k$. Each entry in B^γ is indexed by a $(k+l)$-tuple $(i_1, \ldots, i_{k+l})$. Without loss of generality, we can assume that the k-tuple $(i_{l+1}, \ldots, i_{l+k})$ indexes the entries in the input tensor, and the l-tuple $(i_1, \ldots, i_l)$ indexes the entries in the output tensor. How different equivalence classes translate to different functionalities of the basis functions can be easily observed from the matrix-vector multiplication.

For a given pair of k and l, the IGN basis (Maron et al., 2019b) includes all the partitions of the multiset $\{i_1, \ldots, i_{k+l}\}$, so the size of the basis is equal to the total number of partitions of the multiset. With a multiset of size $k+l$, the number of partitions is bell$(k+l)$. Our basis, on the other hand, only takes a subset of the partitions, therefore, it is smaller. The selection criterion is based on the functionality of the corresponding basis function.

The member equivalence classes can be organized into three categories. In the following, we list the member equivalence classes in each category. Equivalence classes are defined using the general set-builder notation. We use { } to represent a multiset and use ( ) to represent an ordered set (k-tuple). Two ordered sets are equal if and only if their lengths are equal, the corresponding first elements are equal, the corresponding second elements are equal, and so on, i.e., $(i_1, i_2) = (i_3, i_4)$ if and only if $i_1 = i_3$ and $i_2 = i_4$.

Equivalence classes can also be represented by using the enumeration notation. For instance, with $k = l = 2$, the class of $\big\{(i_1, i_2, i_3, i_4) \big| (i_1, i_2) = (i_3, i_4), i_3 \neq i_4\big\}$ can also be written as $\{\{1, 3\}, \{2, 4\}\}$.

(I) Sum of elements along coordinate axes. There are $\binom{k}{1} + \sum_{j=0}^{k-1} \binom{k}{j}$ equivalence classes in this category for output order $l = 0, \ldots, k$. In this category, both case (a) and case (b) require $i_{l+1} \neq i_{l+2} \neq \ldots \neq i_{l+k}$.

(a) For $l = k$. This is summation without dimension reduction. There are $\binom{k}{1} = k$ equivalence classes in $\gamma^{(l)}$ for summation along each axis and then output the sum to different cells in the output tensor:

$$\gamma^{(l)} = \big\{(i_1, \ldots, i_{l+k}) \big| (i_{l+1}, \ldots, i_{l+k}) = (i_1, \ldots, i_{j-1}, i'_j, i_{j+1}, \ldots, i_l),\ j \in [k] \text{ and } i'_j \neq i_j\big\}$$

(b) For $l = k-1, \ldots, 0$. This is a summation with dimension reduction. There are $\sum_{j=0}^{k-1} \binom{k}{j}$ equivalence classes in this category for summation along one axis (for $l = k-1$), up to along k axes, i.e., along the hyperplane spanned by k axes (for $l = 0$), and then output each sum to one cell in the output tensor:

$$\begin{aligned}
&l = k-1, \quad \gamma^{(l)} = \big\{(i_1, \ldots, i_{l+k}) \big| (i_{l+1}, \ldots, i_{l+k}) = (i_1, \ldots, i_{l+j}, \ldots, i_l),\ j \in [k]\big\} \\
&l = k-2, \quad \gamma^{(l)} = \big\{(i_1, \ldots, i_{l+k}) \big| (i_{l+1}, \ldots, i_{l+k}) = (i_1, \ldots, i_{l+j_1}, \ldots, i_{l+j_2}, \ldots, i_l), \\
&\qquad\qquad j_1, j_2 \in [k]\big\} \\
&\vdots \\
&l = 0, \qquad \gamma^{(l)} = \big\{(i_1, \ldots, i_{l+k}) \big| i_{l+1} \neq i_{l+2} \neq \ldots \neq i_{l+k}\big\}
\end{aligned}$$

(II) Replicate diagonals to a lower dimension. There are $\sum_{j=0}^{k-2} \binom{k}{j}$ equivalence classes in this category for output order $l = 1, \ldots, k-1$.

$$\begin{aligned}
&l = k-1, \quad \gamma^{(l)} = \big\{(i_1, \ldots, i_{l+k}) \big| (i_{l+1}, \ldots, i_{l+k}) = (i_1, \ldots, i_j, i'_j, \ldots, i_l),\ j \in [l] \\
&\qquad\qquad \text{and } i'_j = i_j\big\} \\
&l = k-2, \quad \gamma^{(l)} = \big\{(i_1, \ldots, i_{l+k}) \big| (i_{l+1}, \ldots, i_{l+k}) = (i_1, \ldots, i_j, i'_j, \ldots, i''_j, \ldots, i_l),\ j \in [l] \\
&\qquad\qquad \text{and } i'_j = i''_j = i_j\big\} \\
&\vdots \\
&l = 1, \qquad \gamma^{(l)} = \big\{(i_1, \ldots, i_{l+k}) \big| i_l = i_{l+1} = \ldots = i_{l+k}\big\}
\end{aligned}$$

(III) Sum of diagonal elements. There are $\sum_{j=0}^{k-2} \binom{k}{j}$ equivalence classes in this category for output order $l = 0, \ldots, k-2$.

$$
\begin{aligned}
l = k-2, \quad \gamma^{(l)} &= \big\{(i_1, \ldots, i_{l+k}) \big| (i_{l+1}, \ldots, i_{l+k}) = (i_1, \ldots, i_{l+j_1}, \ldots, i_{l+j_2}, \ldots, i_l), \\
&\qquad j_1, j_2 \in [k] \text{ and } i_{l+j_1} = i_{l+j_2} \neq i_j, \ \forall j \in [l] \big\} \\
&\vdots \\
l = 0, \qquad \gamma^{(l)} &= \big\{(i_1, \ldots, i_{l+k}) \big| i_{l+1} = i_{l+2} = \ldots = i_{l+k} \big\}
\end{aligned}
$$

The selected equivalence classes provide either aggregation or dimension reduction, or both. We select the equivalence classes that most efficiently implement the aforementioned functions, and the rest that do not provide additional functionality beyond these are eliminated. For instance, with $k = 2$, and $l = 2$, the equivalence class $\{\{i_1, i_2, i_3\}, \{i_4\}\}$ is included in the IGN basis but not included in our basis, since our basis for $l = 1$ includes an equivalence class $\{\{i_1, i_2\}, \{i_3\}\}$, which serves the same purpose of aggregation. The difference is that our basis also provides dimension reduction by outputting the sums to a length-n vector, while the IGN basis outputs the sums to the diagonal of a tensor of size n^2. It is important to keep the basis small so that fewer trainable parameters are introduced in the neural networks.

Let Γ_k represent the collection of equivalence classes for all linear operators $L : \mathbb{R}^{n^k} \to \mathbb{R}^{n^l}$ with $l = 0, \ldots, k$, we have

$$\Gamma_k = \{\gamma^{(l)} \mid l = 0, \ldots, k\},$$

where $\gamma^{(l)}$ is the set of equivalence classes for output order l. The total dimension $\dim(\Gamma_k)$ is the cardinality of the set Γ_k:

$$
\begin{aligned}
\dim(\Gamma_k) &= \binom{k}{1} + \sum_{j=0}^{k-1} \binom{k}{j} + \sum_{j=0}^{k-2} \binom{k}{j} + \sum_{j=0}^{k-2} \binom{k}{j} \\
&= 3(2^k - 1) - k.
\end{aligned}
$$

The equivalence classes for $k = 2, 3, 4$ are presented in Appendices A.1, A.2, and A.3, respectively. In practice, higher input orders (with $k > 3$) are rarely used. This is not only because the number of basis functions will increase, and hence more trainable parameters, but also due to the memory and computational power required to process high-order tensors of size n^k. Experiment results show that GNNs built with our order-3 bases are sufficient for most applications and also computationally feasible.

It is worth noting that the identity function can be implemented by using a linear operator $L : \mathbb{R}^{n^k} \to \mathbb{R}^{n^k}$ with the equivalence class defined as

$$\gamma = \big\{(i_1, \ldots, i_{k+k}) \big| (i_1, \ldots, i_k) = (i_{k+1}, \ldots, i_{k+k}) \big\},$$

then the corresponding basis function $h_\gamma(X) = B^\gamma X = X$. The unnecessary complexity of multiplying X by B^γ can be avoided by using $h_\gamma(X) = X$ directly in implementation.

3.3 Tensor representation of graphs

Since the neural network operates on input tensors of order k, in what follows, we describe how to represent a graph by using order-k tensors.

A graph $G = (V, E)$ can be represented as an adjacency matrix $A \in \mathbb{R}^{n^2}$, where $A_{i_1,i_2} = 1$ if $(i_1, i_2) \in E$, and otherwise 0. The matrix representation of a graph can be generalized to use tensors with order $k > 2$: Let X be the input tensor that the invariant and equivariant linear layers act on. X is binary (i.e., all entries of $X \in \{0, 1\}$). The input graph $G = (V, E)$ can be represented in the form of $X \in \mathbb{R}^{n^k \times T}$, which has T channels with each channel being an order-k tensor.

The index of an element in the order-k tensor is a k-tuple: $\mathcal{I} = (i_1, i_2, \ldots, i_k)$. Let the notation $[n]$ represent a set $\{1, \ldots, n\}$. We use $k - 1$ channels to encode the link structure according to E as follows:

$$X_{\mathcal{I},r} = 1 \text{ if and only if } (i_r, i_{r+1}) \in E, \text{ for } r \in [k-1].$$

The first channel represents the relation between i_1 and i_2, and the second channel represents the relation between i_2 and i_3, and so on. A total of $k - 1$ channels are needed for encoding E. This encoding scheme applies to not only undirected graphs but also directed graphs. For directed graphs, X is no longer symmetric.

It is noteworthy that this linear transformation from adjacency matrix A to order-k tensor X is also permutation-equivariant. It is equivalent to having a linear operator $L : \mathbb{R}^{n^2} \to \mathbb{R}^{n^k}$ applied to each channel, which performs the multiplication of a basis element B of dimension n^{2+k} with a matrix A of dimension n^2, and result in a tensor X of dimension n^k. The support in the basis element is defined based on the equality pattern in the m-tuple index (with $m = 2 + k$). The proof follows the same argument in section 3.1 that the permutation operator g does not change the equivalence class of the m-tuple, i.e., $L(g \cdot A) = g \cdot L(A)$. Therefore, the initialization step can be considered as an equivariant linear transformation from $\mathbb{R}^{n^2}$ to $\mathbb{R}^{n^k}$.

Node features and edge features The binary representation of a graph can be extended to include additional f_v node features and f_e edge features. Let F_V and F_E represent the set of node features and edge features, respectively. While the first $k - 1$ channels encode the link structure of the graph according to E, additional node features and edge features can use additional channels:

- If there are additional f_e edge features, there will be additional $f_e(k - 1)$ channels representing these edge features. The encoding of the edge features is similar to the encoding of the adjacency matrix, with each feature corresponding to $k - 1$ channels.
- If there are f_v node features, there will be additional f_v channels to represent the node features. Each channel is an order-k tensor, with the diagonal positions representing the node features, and off-diagonal positions being zero.

With node and edge features, the total number of channels $T = (k - 1)(1 + f_e) + f_v$. The encoding of input graph $G = (V, E, F_V, F_E)$ in the form of $X \in \mathbb{R}^{n^k}$ can also be viewed as the operation of a linear operator $L : \mathbb{R}^{n^2 \times (1+f_e+f_v)} \to \mathbb{R}^{n^k \times ((k-1)(1+f_e)+f_v)}$.

For encoding E and edge features:

$$L(A)_{\mathcal{I},t(r,w)} = A_{i_r,i_{r+1},w}, \;\; r \in [k-1], \; w \in [1+f_e],$$

where the linear index $t(r,w) = (w-1)(k-1)+r$ identifies which tensor, and $\mathcal{I}$ identifies which entry in the tensor.

For encoding node features:

$$L(A)_{\mathcal{I},(k-1)(1+f_e)+w} = \begin{cases} A_{j,j,1+f_e+w}, \; j \in [n], w \in [f_v], & \text{if } \mathcal{I} = (j,\dots,j); \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

This is a more efficient encoding scheme than the previous work (Maron et al., 2019b,a), since it uses only $k-1$ channels for each of the edge-related encodings, and uses only one channel for each node feature, instead of using k^2 channels as in Maron et al. (2019b,a).

3.4 Difference from k-IGN in Maron et al. (2019b)

The differences with previous work (Maron et al., 2019b) can be summarized in two major points, which may contribute to the improved computational feasibility of the proposed basis.

- Fewer number of equivalence classes are used for the linear basis. Previous work (Maron et al., 2019b) has a much larger basis. It uses a maximal set of equivalence classes in the sense that they include all the possible partitions of the set $\{1,2,\dots,k+l\}$, and there are no equality patterns of the input and output indices that are not included in its collection. The number of equivalence classes for an equivariant layer $L : \mathbb{R}^{n^k} \to \mathbb{R}^{n^l}$ is $\text{bell}(k+l)$. Despite its exhaustiveness, not all equivalence classes are necessary, as many of them are not useful in training the neural network, which results in $w_\gamma \to 0$ for the unused equivalence class.

  Given a specific output order l, our basis is a subset of the maximal set. The total dimension of our bases for $l = 0,\dots,k$ is bounded by $\dim(\Gamma_k)$, and $\dim(\Gamma_k) \ll \sum_{l=0}^{k} \text{bell}(k+l)$. In fact, $\dim(\Gamma_k) \ll \text{bell}(2k)$ for $k > 2$, which is the dimension for only one equivariant layer $L : \mathbb{R}^{n^k} \to \mathbb{R}^{n^k}$ in k-IGN (Maron et al. (2019b)).

  Even though our basis uses only a subset of the maximal set, it is not losing discriminative power when it comes to distinguishing graphs. We next prove that our basis has the same expressive power as the maximal set measured by the standard graph isomorphism testing. In addition, it also preserves the orthogonality, invariance, and equivariance properties.

- A Fewer number of tensors are used for initial tensor representation. To represent a graph using order-k tensors, we use $k-1$ tensors for representing the adjacency matrix or an edge feature, while k-IGN uses k^2 tensors for the same purpose. Note that our representation does not require the adjacency matrix A to be symmetric, so it can represent both undirected and directed graphs.

  The reason why we can use only $k-1$ tensors is that we use the multi-index of the order-k tensor as an ordered k-tuple, so the r-th tensor encodes the relationship of

vertex i_r to vertex i_{r+1}, with $r \in [k-1]$. In doing so, the order given in the multi-index is being considered. k-IGN, on the other hand, uses the multi-index as an unordered set, so it needs $k \times k$ tensors to encode the pair-wise relationship between vertex i_r and vertex i_s, with $r, s \in [k]$.

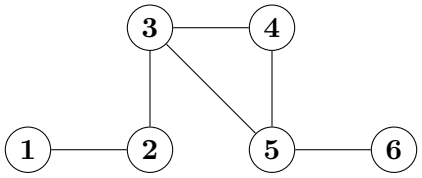


Figure 1: An example for tensor representation of graphs.

For the 6-node graph in Figure 1, using order-3 tensors we only need $k-1=2$ tensors to encode the graph adjacency matrix A, while k-IGN needs $k^2 = 9$ tensors. Let $\mathcal{I} = (i_1, i_2, i_3)$ represent the 3-tuple index for entries in the tensor. We consider two examples: $\mathcal{I}_1 = (2,3,4)$ and $\mathcal{I}_2 = (1,5,6)$.

Ours: $X_{\mathcal{I},r} = A_{i_r,i_{r+1}},\ r \in [k-1]$	k**-IGN:** $X_{\mathcal{I},r,s} = A_{i_r,i_s},\ r,s \in [k]$
$X_{(2,3,4),1} = 1$	$X_{(2,3,4),1,1} = 0,\ X_{(2,3,4),1,2} = 1,\ X_{(2,3,4),1,3} = 0$
$X_{(2,3,4),2} = 1$	$X_{(2,3,4),2,1} = 1,\ X_{(2,3,4),2,2} = 0,\ X_{(2,3,4),2,3} = 1$
	$X_{(2,3,4),3,1} = 0,\ X_{(2,3,4),3,2} = 1,\ X_{(2,3,4),3,3} = 0$
$X_{(1,5,6),1} = 0$	$X_{(1,5,6),1,1} = 0,\ X_{(1,5,6),1,2} = 0,\ X_{(1,5,6),1,3} = 0$
$X_{(1,5,6),2} = 1$	$X_{(1,5,6),2,1} = 0,\ X_{(1,5,6),2,2} = 0,\ X_{(1,5,6),2,3} = 1$
	$X_{(1,5,6),3,1} = 0,\ X_{(1,5,6),3,2} = 1,\ X_{(1,5,6),3,3} = 0$

4. Properties of the linear bases

We show the proposed linear bases satisfy the orthogonality, invariance, and equivariance properties. In addition, the invariance and equivariance properties of the linear bases also lead to the permutation invariant property of the neural network.

4.1 Orthogonality property

Given a linear layer L that maps from input order k to output order l, the set of basis functions $\{B^\gamma X\}$ are orthogonal to each other since their supports have no overlap, i.e., B^{γ_i} and B^{γ_j} have no overlap in the non-zero elements for $i \neq j$. We can easily prove the orthogonal property since our basis is a subset of the maximal set in Maron et al. (2019b). The maximal set is based on set partitions and there are no overlaps between the partitions.

Theorem 1 *For any $l \in \{0, \ldots, k\}$, tensors in the family of $B^{\gamma^{(l)}}$ form an orthogonal basis for the linear layer $L : \mathbb{R}^{n^k} \to \mathbb{R}^{n^l}$.*

Proof. $\gamma^{(l)} = \{\gamma_1^{(l)}, \gamma_2^{(l)}, \ldots\}$ is a set of pairwise disjoint subsets with each representing an equivalence class. Since the supports of elements in $B^{\gamma^{(l)}}$ are pairwise disjoint, $B^{\gamma^{(l)}}$ forms an orthogonal basis for $L : \mathbb{R}^{n^k} \to \mathbb{R}^{n^l}$. □

4.2 Invariance and equivariance property

Let X be a tensor representing the graph. Using the definition from Maron et al. (2019b), a linear layer $L : \mathbb{R}^{n^k} \to \mathbb{R}^{n^l}$ is called an equivariant layer if it satisfies

$$L(g \cdot X) = g \cdot L(X),$$

and a linear layer $L : \mathbb{R}^{n^k} \to \mathbb{R}$ is called an invariant layer if it satisfies

$$L(g \cdot X) = L(X).$$

That is, for the equivariant layer L, permutation on the graph is equivalent to permutation on the output of L; for the invariant layer L, permutation on the graph will not change the output.

We now show that the linear layers consisting of the proposed basis functions are permutation invariant or equivariant.

Theorem 2 *For any $l \in \{0, \ldots, k\}$, and any permutation operator $g(\cdot)$ acting on n elements, the linear operator $L : \mathbb{R}^{n^k} \to \mathbb{R}^{n^l}$ in equation (1) is permutation invariant for $l = 0$ and equivariant for $l \geq 1$:*

$$L(g \cdot X) = L(X), \text{ for } l = 0;$$
$$L(g \cdot X) = g \cdot L(X), \text{ for } l \geq 1.$$

Proof. See Appendix A.4 for the proof of Theorem 2.

Permutation invariant network A typical network F is a chain of multiple layers. If each layer implements a mapping function, F can be written as a composite function:

$$F = m \circ L_d \circ \ldots \circ L_1, \tag{3}$$

where m is typically a multilayer perceptron (MLP), and L is defined above.

Theorem 2 leads to the following corollary:

Corollary 3 *Let F be a graph neural network composed of linear layers based on the proposed basis following the architecture of (3). Given two isomorphic graphs $G = (V, E)$ and $G' = (V', E')$, F outputs the same result: $F(G) = F(G')$.*

Proof. We show that a *sufficient condition* for F being permutation invariant is that each $L_i : \mathbb{R}^{n^{k_i}} \to \mathbb{R}^{n^{l_i}}$ (for $i < d$) is an equivariant layer, and $L_d : \mathbb{R}^{n^{k_d}} \to \mathbb{R}$ is an invariant layer.

The deep neural network F is invariant since the equivariance property is propagated from L_1 to L_{d-1}. The last linear layer L_d makes it permutation invariant. Following each

linear layer, there is an activation function σ, i.e., $F = m \circ (\sigma \circ L_d) \dots (\sigma \circ L_1)$, Since σ is an entry-wise activation function, it does not change the node order, and therefore it allows the invariance property to propagate through the network. The MLP m is applied to the output of L_d, which is already permutation invariant:

$$\begin{aligned} F(g \cdot X) &= m(L_d(L_{d-1} \dots L_1(g \cdot X))) \\ &= m(L_d(L_{d-1} \dots g \cdot L_1(X))) \\ &= m(L_d(g \cdot L_{d-1} \dots L_1(X))) \\ &= m(L_d(L_{d-1} \dots L_1(X))) \\ &= F(X) \end{aligned}$$

5. The expressive power of graph neural networks

In this section, we assess the discriminative power of graph neural networks built with our basis by using k-WL graph isomorphism tests. For preliminaries of graph isomorphism tests and k-WL/k-FWL algorithms, see Appendix A.5.

The k-WL algorithm is an iterative algorithm for graph-isomorphism testing (Weisfeiler and Leman, 1968; Grohe, 2017). Given a graph, the algorithm outputs its color distribution upon termination. In each iteration, the algorithm uses an injective hash function to update the colors of the "nodes". In the k-WL algorithm, the "nodes" are the k-tuples. When the color update stabilizes, the algorithm terminates and outputs its color distribution. To determine if two graphs G and H are isomorphic, we need to apply k-WL on each graph separately, generating the color distribution of each, and then based on whether the color distributions are the same or different, conclude if the two graphs are isomorphic.

A network F is said to have k-WL expressive power if F can distinguish any pair of non-isomorphic graphs that are distinguishable by the k-WL algorithm. In other words, a network with k-WL expressive power is at least as powerful as the k-WL algorithm in graph isomorphism tests.

Using constructive proof, we will show that the graph neural networks using the proposed order-k linear basis have k-WL expressive power, and then we will advance the discriminative power to k-FWL by proposing a new building block.

5.1 k-WL discriminative power

To represent a graph, an input tensor $X \in \mathbb{R}^{n^k}$ has order $k \geq 2$, i.e., the lowest order for an input tensor is an adjacency matrix. In this section, we demonstrate that the order-k neural networks using the proposed basis, have at least k-WL expressive power.

Theorem 4 *Given two non-isomorphic graphs $G = (V, E)$ and $G' = (V', E')$, if G and G' can be distinguished by the k-WL algorithm, then there exists an order-k network F so that $F(G) \neq F(G')$.*

Proof. We use inductive proof to show that for a given graph $G = (V, E)$, the neural network F would produce the same color distribution as the k-WL algorithm.

Initialization: In the k-WL algorithm, the initial color assignment of a node is based on the atomic type of the k-tuple (Morris et al., 2019; Grohe, 2017); in order-k GNN, each entry of the tensor is indexed by a k-tuple and the entry is initialized with a color corresponding to its atomic type. One-hot encoded binary vector can be used for representing multiple colors.

We next show that each color update step in the k-WL algorithm is equivalent to a linear layer constructed with our basis.

As shown in previous work (Maron et al., 2019a), a three-step procedure is what it takes to implement an iteration in the k-WL algorithm. The first step is to find the colors of the k-WL neighborhoods of each node. Since the colors of each k-WL neighborhood are represented as a multiset, this amounts to using a linear operator to count the number of nodes in each distinct color. The second step is to concatenate a node's color with its k neighborhoods' multisets of colors to generate the input to the injective hash function. The third step is to apply the hash function to update the colors of the nodes, thus completing one color-updating step.

It is sufficient to show that there is a one-to-one correspondence between the linear operator in the first step and a linear layer $L : \mathbb{R}^{n^k} \to \mathbb{R}^{n^{k-1}}$ based on our basis. The concatenation step is trivial. The injective hash function that performs color assignment in the third step is approximated by an MLP (see Maron et al., 2019a, for more details).

The k-WL neighborhood is defined as an ordered k-tuple, with each element being an unordered multiset of n elements. Let $v = (i_1, \ldots, i_k)$ be the node to be updated, and $N_j(v)$ be the j-th neighborhood of node v, for $j \in [k]$,

$$N_j(v) = \{(i_1, \ldots, i_{j-1}, w, i_{j+1}, \ldots, i_k) \mid w \in [n]\}.$$

Then the injective hash function outputs the new color for node v in the l-th iteration:

$$c_v^l = \text{hash}\left(c_v^{l-1}, \left(\{c_u^{l-1} \mid u \in N_j(v)\}, j \in [k]\right)\right)$$

Since $N_j(v)$ is an unordered multiset of size n, with each element in the set being a "node" given by a k-tuple index, the colors of the nodes in $N_j(v)$ can be described by color counts, e.g., 4 blues, 2 reds, etc. A linear operator defined on our basis $L^{(j)} : \mathbb{R}^{n^k} \to \mathbb{R}^{n^{k-1}}$ that takes the sum of elements along the j^{th} axis ($j \in [k]$) of the input tensor X exactly outputs the colors of the j^{th} neighborhood of a node as defined in the k-WL algorithm:

$$Y_j = L^{(j)}(X) = \sum_{w=1}^{n} X_{i_1,\ldots,i_{j-1},w,i_{j+1},\ldots,i_k},$$

where the summation over index w amounts to counting the colors in the j-th neighborhood $N_j(v)$.

The concatenation of $(X, Y_1, \ldots, Y_k)$ is then used as input to the hash function in the k-WL algorithm. For an injective hash function, the output is uniquely determined by the input. An MLP is used to approximate the hash function that maps k-tuple nodes to colors. Thus, we established the equivalence of an update step in the k-WL algorithm and a linear layer of the neural network.

If it takes T iterations in the k-WL algorithm until convergence, then the neural network can use T equivariant linear layers $L_i : \mathbb{R}^{n^k} \to \mathbb{R}^{n^{k-1}}$, for $i = 1, \ldots, T$.

Consider a k-tuple node $v = (i_1, \ldots, i_k)$ in k-WL and an entry at $\mathcal{I} = (i_1, \ldots, i_k)$ of the input tensor of the network F. Initially they have the same "color", and we denote this by $c_v^0 = c_{\mathcal{I}}^0$.

The correspondence between an iteration of k-WL and a linear layer of the neural network indicates that if $c_v^l = c_{\mathcal{I}}^l$, then $c_v^{l+1} = c_{\mathcal{I}}^{l+1}$, for $l = 0, \ldots, T-1$.

After T iterations, the k-WL algorithm produces the same colors as F outputs.

Now we extend to the case when the graph has a non-empty set of node features. Node features are handled by using one channel per pair of input-output features in the neural network, and the above equivalence between an update step in k-WL and a linear layer in the neural network holds for each channel.

Finally, the k-WL algorithm returns the colors of all nodes upon termination, which is represented as a multiset, essentially giving the number of nodes in each distinct color. To get the histogram of colors, we use a summing invariant layer h, i.e.,

$$F = h \circ (m \circ L_T) \circ \ldots \circ (m \circ L_1).$$

If two graphs generate the same color distributions, the two graphs are indistinguishable. A follow-up procedure will compare the color distributions from two graphs and conclude the isomorphism test. The same procedure is applied to the two histograms generated by the GNN.

We conclude that if the k-WL algorithm can distinguish between two non-isomorphic graphs G and G', then there exists a neural network F such that $F(G) \neq F(G')$. □

5.2 k-FWL discriminative power

We construct a family of networks that replaces the linear layer $L : \mathbb{R}^{n^k} \to \mathbb{R}^{n^{k-1}}$, which implements one update step of a k-WL algorithm, with a building block as shown in Figure 2. The building block includes an equivariant layer $L : \mathbb{R}^{n^k} \to \mathbb{R}^{n^{k+1}}$, an element-wise Boolean function, and another equivariant layer $L : \mathbb{R}^{n^{k+1}} \to \mathbb{R}^{n^k}$. We call such a network type-b network or GNN-b, and call the previous network type-a network or GNN-a. We show that GNN-b networks with input order-k can have the expressive power of the k-FWL algorithm.

Construction of type-b networks Figure 2 shows the building block for $k = 2$. The first linear layer $\mathbb{R}^{n^2} \to \mathbb{R}^{n^3}$ does the following:

$$L_1(X)_{w,i,j} = X_{w,j}; \text{ and } L_2(X)_{w,i,j} = X_{i,w}.$$

Let $A_1 = L_1(X)$ and $A_2 = L_2(X)$. Then the element-wise Boolean function $f_{j'} : \mathbb{R}^{n^3} \times \mathbb{R}^{n^3} \to \mathbb{R}^{n^3}$, for $j' = 1, 2, 3, 4$, are defined as:

$$\begin{aligned} Z_1 &= f_1(A_1, A_2) = A_1 \vee A_2 \oplus 1 \\ Z_2 &= f_2(A_1, A_2) = 1 \oplus A_1 \wedge A_2 \\ Z_3 &= f_3(A_1, A_2) = 1 \oplus A_2 \wedge A_1 \\ Z_4 &= f_4(A_1, A_2) = A_1 \wedge A_2 \end{aligned}$$

where $\vee$ is logical OR operator, $\wedge$ is logical AND operator, and $\oplus$ is logical exclusive OR (XOR) operator. Each Boolean function $f_{j'}$ turns on exactly one of the possible values of

the 2-tuple. For instance, $Z_1 = 1$ if and only if the input is (0,0); $Z_2 = 1$ if and only if the input is (0,1), and so on.

The second linear layer $L : \mathbb{R}^{n^3} \to \mathbb{R}^{n^2}$ uses the basis as defined in section 3.1 by summing along one axis.

$$Y_{i,j,j'} = \sum_{w=1}^{n} Z_{w,i,j,j'}, \quad j' = 1, 2, 3, 4, \tag{4}$$

where $Y_{i,j,j'}$ is an entry in $Y_{j'}$ indexed by a 2-tuple index (i, j), and $Z_{w,i,j,j'}$ is an entry in $Z_{j'}$ indexed by a 3-tuple index (w, i, j), for $j' = 1, 2, 3, 4$.

Since $f_{j'}$ are element-wise functions, the implementation of a type-b building block does not need to use higher order tensors As and Zs. The mapping from X to $Y_1, \ldots, Y_4$ amounts to the following operation:

$$Y_{i,j,j'} = \sum_{w=1}^{n} f_{j'}(X_{w,j}, X_{i,w}), \text{ for } j' = 1, 2, 3, 4. \tag{5}$$

Equations (4) and (5) are equivalent.

If the color representation uses c bits, then we use c channels in the neural network with the above functions acting on each channel. For an order k type-b building block, the input tensor $X \in \mathbb{R}^{n^k \times c}$, $A_1, \ldots, A_k \in \mathbb{R}^{n^{k+1} \times c}$, $Z_1, \ldots, Z_{2^k} \in \mathbb{R}^{n^{k+1} \times c}$, and $Y_1, \ldots, Y_{2^k} \in \mathbb{R}^{n^k \times c}$. If we leave out the intermediate steps, the mapping from X to $Y_1, \ldots, Y_{2^k}$ amounts to the following operation, similar to equation (5),

$$Y_{i_1,\ldots,i_k,j'} = \sum_{w=1}^{n} f_{j'}(X_{w,i_2,\ldots,i_k}, X_{i_1,w,i_3,\ldots,i_k}, \ldots, X_{i_1,\ldots,i_{k-1},w}), \text{ for } j' = 1, \ldots, 2^k, \tag{6}$$

where $f_1, \ldots, f_{2^k}$ are the element-wise Boolean functions. Each Boolean function $f_{j'}$ turns on exactly one of the possible values of the k-tuple. i.e.,

$$\begin{aligned} f_1(A_1, \ldots, A_k) &= A_1 \vee A_2 \vee \ldots A_k \oplus 1 \\ &\vdots \\ f_{2^k}(A_1, \ldots, A_k) &= A_1 \wedge A_2 \wedge \ldots A_k \end{aligned}$$

Discriminative power of type-b networks

Theorem 5 *Given two non-isomorphic graphs $G = (V, E)$ and $G' = (V', E')$, if G and G' can be distinguished by the k-FWL algorithm, then there exists an order-k GNN-b network F^b so that $F^b(G) \neq F^b(G')$.*

Proof. The k-FWL algorithm is also an iterative algorithm for graph isomorphism testing. In each update step, the algorithm uses an injective hash function to update the color of a node. However, it differs from the k-WL algorithm in its updating rule.

Let $v = (i_1, \ldots, i_k)$ be the node to be updated. The input to the hash function of the k-FWL algorithm is the node's color, concatenated with the colors of nodes in its

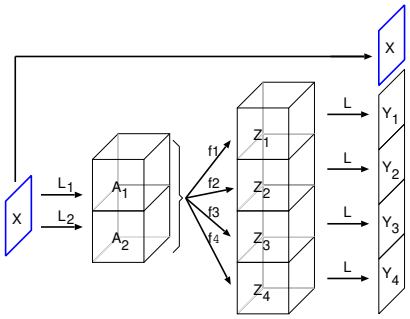


Figure 2: A building block for GNN-b. The illustration is for $k = 2$, $c = 1$.

neighborhood. The k-FWL neighborhood is an unordered multiset of n elements $\{N_j^F(v) \mid j \in [n]\}$ with each $N_j^F(v)$ being an ordered set of k elements (k-tuple).

$$N_j^F(v) = ((j, i_2, \ldots), (i_1, j, \ldots), \ldots, (\ldots, i_{k-1}, j))$$

Then the injective hash function outputs the new color for node v:

$$c_v^l = \text{hash}\left(c_v^{l-1}, \left\{c_u^{l-1} \mid u \in N_j^F(v), j \in [n]\right\}\right)$$

We now demonstrate the correspondence between an update step of the k-FWL algorithm and a building block in Figure 2. For an entry (i, j) in the input tensor X, L_1 collects colors from the same column (the j-th column), and L_2 collects the colors from the same row (the i-th row), and replicate the information to a tensor of order $k+1$. This step creates k-tuples, with each k-tuple indexed by w, for $w \in [n]$.

Subsequently, the element-wise Boolean functions $f_{j'}$ get the possible color patterns of the k-tuples and put them in different categories.

The second linear layer L aggregates the total number of occurrences in each category by summing the n unordered elements.

The output of the building block in Figure 2 is the concatenation of (X, Y_1, Y_2, Y_3, Y_4) for $k = 2$. For any $k \geq 2$, the output of the building block is the concatenation of $(X, Y_1, \ldots, Y_{2^k})$, which is exactly the neighborhood definition in the k-FWL algorithm.

Thus, we established the equivalence between an update step in the k-FWL algorithm and a building block of a type-b network. Similarly, if it takes T^b iterations for the k-FWL algorithm to converge, it also needs T^b building blocks in the neural network F^b. There will be an MLP m after each building block for color assignment. Using the same initial color assignment as in k-WL, inductive proof leads to the conclusion that the output from the type-b network is the same as the output from the k-FWL algorithm.

Similar to the proof of Theorem 4, there is an invariant layer h after the T^b blocks, so we have

$$F^b = h \circ (m \circ B_{T^b}) \circ \ldots \circ (m \circ B_1).$$

Using inductive proof, we can demonstrate that if the initial color assignment in the k-FWL is the same as in the network F^b, then after T^b iterations, the k-FWL will produce the same output as F^b. This concludes that if the k-FWL algorithm can differentiate between

two non-isomorphic graphs G and G', then there must be a neural network F^b such that $F(G) \neq F(G')$.

5.3 Identifying isomorphic graphs

To fully characterize the expressive power GNN-b, it is also important to check its ability to recognize isomorphic graphs.

Theorem 6 *Given two isomorphic graphs $G = (V, E)$ and $G' = (V', E')$, the type-b network F^b constructed in Theorem 5 outputs the same result: $F^b(G) = F^b(G')$.*

Proof. Since G and G' are isomorphic graphs, G' can be represented as $G' = g \cdot G$, where g is a permutation operator.

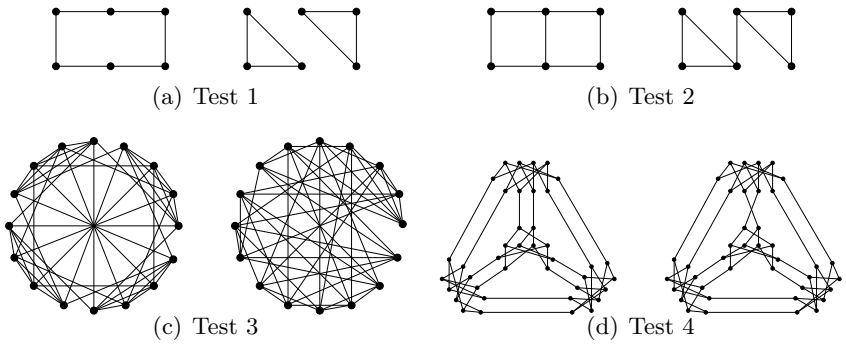
A type-b network $F^b = h \circ B_{T^b} \circ \ldots \circ B_1$ is permutation invariant since it is a chain of the building blocks shown in Figure 2. Each building block is a chain of the equivariant linear layers and Boolean functions. The equivariance property can propagate through the building blocks since the element-wise functions $f_{j'}$ do not change the equivariance property as they act on elements. Therefore, $F^b(G') = F^b(g \cdot G) = F^b(G)$. □

Corollary 7 *Order-k GNN-a has the expressive power of the k-WL algorithm and order-k GNN-b has the expressive power of the k-FWL algorithm in graph isomorphism testing.*

Since k-FWL is equivalent to $(k+1)$-WL in graph isomorphism testing for $k \geq 2$ (Grohe, 2017; Cai et al., 1992), we conclude that order-2 GNN-b has 3-WL expressiveness in graph isomorphism testing.

5.4 Graph isomorphism testing examples

Theorems 4 and 5 indicate that in terms of distinguishing non-isomorphic graphs, GNN-a is at least as discriminative as k-WL, and GNN-b is at least as discriminative as k-FWL. Next, we verify their discriminative power using graph isomorphism tests (see Figure 3).



(a) Test 1 (b) Test 2

(c) Test 3 (d) Test 4

Figure 3: (a) and (b), non-isomorphic graphs not distinguishable by 2-WL test; (c) and (d), non-isomorphic graphs not distinguishable by 3-WL test.

Test 1 is a pair of 2-regular graphs; Test 2 is tweaked from test 1 by adding an edge to the nodes in the middle; Test 3 is a pair of strongly regular graphs in the family of SR(16,6,2,2), which means that they each have 16 nodes with node degree 6, every two adjacent nodes have two common neighbors, and every two non-adjacent nodes have two common neighbors. The graph on the right (Shrikhande graph) has a maximum clique of size 3, and the graph on the left (Rooks' 4×4 graph) has a maximum clique of size 4. Test 3 has been used in Bouritsas et al. (2023) and Bodnar et al. (2021b). Test 4 from Grohe (2017) is a pair of 3-regular graphs. It is known that 2-WL cannot distinguish the pair of graphs in test 1 and test 2; 3-WL cannot distinguish the pair of graphs in test 3 and test 4 (Grohe, 2017).

Table 1: Graph isomorphism tests results. Y: succeeded; N: failed.

	Test 1	Test 2	Test 3	Test 4
2-WL test	N	N	N	N
3-WL test	Y	Y	N	N
2-FWL test	Y	Y	N	N
PPGN	Y	Y	N	N
Order-2 GNN-a	N	N	N	N
Order-2 GNN-b	Y	Y	N	N

Order-2 GNN-a and order-2 GNN-b are constructed following the description in the proofs for Theorem 4 and Theorem 5, respectively. Testing results show that order-2 GNN-a has the same performance as 2-WL, and order-2 GNN-b has the same performance as 3-WL. PPGN (Maron et al., 2019a) also has 3-WL power, but our models are significantly faster due to having fewer trainable parameters and not requiring expensive matrix multiplication operations.

6. Experiments

In this section, we assess the performance of the proposed type-a and type-b GNNs using real-world datasets, considering both classification and regression tasks.

6.1 GNN models

We implement order-2 and order-3 GNN models as described in section 3. The implementation is carried out in TensorFlow framework using Keras neural network API. The experiments were carried out using the Open Science Grid with AMD EPYC 7F52 16-Core 3.50 GHz Processor.

We use the order-2 GNN-b and order-3 GNN-a to learn the link structure. Node features are represented as $\mathbb{R}^n$ vectors and are combined with the output of equivariant layer $L : \mathbb{R}^{n^k} \to \mathbb{R}^n$ before entering the next invariant layer.

6.2 Classification tasks

We show the graph classification performance on real-world datasets and compare our methods (order-2 GNN-b and order-3 GNN-a) with other baseline methods from the literature: WEGL (Kolouri et al., 2021), DGK (Yanardag and Vishwanathan, 2015), DGCNN (Zhang et al., 2018), 2-IGN (Maron et al., 2019b), PPGN (Maron et al., 2019a), GIN (Xu et al., 2019), DropGIN (Papp et al., 2021), 1-2-3 GNN (Morris et al., 2019), GFN (Chen et al., 2019a), InfoGraph (Sun et al., 2019), BC + Capsules (Mallea et al., 2019), hGANet (Gao and Ji, 2019), CCN (Kondor et al., 2018), CIN Bodnar et al. (2021a), GSN (Bouritsas et al., 2023), and SIN (Bodnar et al., 2021b).

The TUDatasets include MUTAG, PTC, PROTEINS, NCI1, NCI109, COLLAB, IMDB-B, and IMDB-M selected from Yanardag and Vishwanathan (2015). The first five datasets include protein structures and chemical compounds from bioinformatics datasets, and the last three datasets are from social networks.

6.2.1 Results

Classification accuracy The results on classification accuracy are reported in Table 2. We follow the protocol of Xu et al. (2019) to perform a standard 10-fold cross-validation with data splitting protocol from Zhang et al. (2018). Validation accuracies averaged over 10 validation folds from the best-performing epoch are reported. Hyper-parameters we tuned for these datasets are: batch size $\in \{64, 128\}$, learning rate $\in \{0.00064, 0.00128, 0.00256\}$, learning rate decay $\in \{0.6, 0.7, 0.8\}$, and learning rate decay of every 40 epochs is used. Hyper-parameter tuning is done within cross-validation with one training fold used for hyper-parameter searching.

Accuracies for baseline methods are cited from the literature. Out of the eight datasets, our methods achieved the highest accuracy on five datasets and the second highest on two datasets.

Computation times We use PPGN (Maron et al., 2019a) as the baseline to compare computational complexity since PPGN is the winner among other algorithms from the literature. We report the training time per epoch and the number of training parameters in the linear layers in Table 3. Our method order-3 GNN-a has a notable advantage since the number of training parameters is significantly reduced compared to PPGN and hence faster training time. Results are reported using the first four datasets as examples, and other datasets have the same conclusion.

6.3 Regression tasks

6.3.1 QM9 dataset

For regression task, we first test our graph neural network models on a standard graph learning benchmark dataset called QM9 (Ramakrishnan et al., 2014). The dataset consists of 134K organic molecules with sizes ranging from 4 to 29 atoms. The task is to predict 12 real-valued physical quantities for each molecule. We compare our models (order-2 GNN-b, and order-3 GNN-a) with other baseline methods. Results for 1-GNN and 1-2-3 GNN are from Morris et al. (2019), and PPGN results are from Maron et al. (2019a).

Table 2: Graph classification accuracy (%) on TUDatasets.

	MUTAG	PTC	PROTEINS	NCI1	NCI109	COLLAB	IMDB-B	IMDB-M
WEGL	88.3	67.5	76.5	-	-	78.6	72.0	48.5
DGK	87.4	60	75.6	80.3	80.3	73.1	66.9	44.5
DGCNN	85.8	58.6	75.5	74.4	-	73.7	70.0	47.8
2-IGN	84.61	59	75.19	74.3	72.8	78.3	72.0	48.7
PPGN	90.6	66.2	77.2	83.2	82.2	81.3	73	50.4
GIN	89.4	64.6	76.2	82.7	-	80.2	75.1	52.3
1-2-3 GNN	86.1	60.9	75.5	76.2	-	-	74.2	49.5
GFN	90.84	-	76.46	82.7	-	80.4	73.3	51.2
DropGIN	90.4	66.3	76.3	-	-	-	75.7	51.4
InfoGraph	89.01	61.65	-	-	-	-	73.0	49.7
BC + Capsules	88.9	69	74.1	65.9	58.0	-	-	-
hGANet	90	65.02	**78.65**	-	-	77.5	-	49.1
CCN	91.64	70.62	-	76.27	75.54	-	-	-
CIN	92.7	68.2	77	83.6	**84**	-	75.6	52.7
GSN	92.2	68.2	76.6	83.5	-	**85.5**	**77.8**	**54.3**
SIN	-	-	76.4	82.7	-	-	75.6	52.4
Order-2 GNN-b	92	**72.4**	73.9	80	81	78.5	72.4	51
Order-3 GNN-a	**94.4**	72.1	77.8	**85**	83.4	80.7	74.9	52.5
Rank	1^{st}	1^{st}	2^{nd}	1^{st}	2^{nd}	3^{rd}	6^{th}	3^{rd}

Table 3: Top: training time per epoch (in seconds); Bottom: number of parameters in linear layers.

	MUTAG	PTC	PROTEINS	NCI1
PPGN	10.56	44.8	865	900
Order-3 GNN-a	9	39.1	223	238
PPGN	2409600	2427600	2404800	2445600
Order-3 GNN-a	196500	211150	136500	214000

The experiment setup is the same as in Maron et al. (2019a). The dataset is randomly split with 80% training data, 10% validation data, and 10% test data. Hyper-parameters are searched by experimenting on the validation set, with learning rate in the set $\{0.001, 0.0001\}$ and batch size in the set $\{64, 128\}$.

We trained a separate neural network model for each quantity and minimized the mean squared error during model training. The results reported from the baseline method PPGN Maron et al. (2019a) are also from using an independent network for each quantity.

The mean absolute error (MAE) on test data is reported in Table 4. Our models achieved the lowest error on 6 out of the 12 quantities. Moreover, the training time per epoch is also reported in Table 5. PPGN is the winner of the four baseline methods, which achieved the

lowest error on 4 out of the 12 quantities, however, our methods demonstrated significant time reduction from PPGN as shown in Table 5.

Table 4: Regression on QM9 dataset. Mean Absolute Error (MAE) is reported.

Target	1-GNN	1-2-3 GNN	PPGN	Order-2 GNN-b	Order-3 GNN-a	Rank
μ	0.493	0.476	**0.0934**	0.120	0.112	2^{nd}
α	0.78	**0.27**	0.318	0.463	0.369	3^{rd}
ϵ_{homo}	0.00321	0.00337	0.00174	0.00224	**0.00172**	1^{st}
ϵ_{lumo}	0.0035	0.00351	0.0021	**0.0020**	**0.0020**	1^{st}
Δ_ϵ	0.0049	0.0048	0.0029	**0.0025**	0.0033	1^{st}
$\langle R^2 \rangle$	34.1	22.9	**3.78**	4.78	5.10	2^{nd}
$ZPVE$	0.00124	**0.00019**	0.000399	0.000452	0.00036	2^{nd}
U_0	2.32	0.0427	**0.022**	0.039	0.026	2^{nd}
U	2.08	0.111	0.0504	0.0521	**0.0478**	1^{st}
H	2.23	0.0419	0.0294	0.0311	**0.0281**	1^{st}
G	1.94	0.0469	**0.024**	0.029	**0.024**	1^{st}
C_v	0.27	**0.0944**	0.144	0.152	0.144	2^{nd}

Table 5: Training time (in seconds) per epoch on QM9 dataset.

PPGN	Order-2 GNN-b	Order-3 GNN-a
3300	70	178

6.3.2 Zinc12k dataset

We further test our model on a large-scale molecular dataset for a regression task. The dataset is called ZINC 12k, which has been used for benchmarking GNNs for expressivity by many (e.g., Sterling and Irwin, 2015; Dwivedi et al., 2023; Gómez-Bombarelli et al., 2018). The dataset consists of 12k molecules with a graph size of up to 29 atoms. There are 21 atom types and three bond types, which we use as node features and edge features.

The task is to predict the constrained solubility. We compare our model with other models reported in Table 3 of Bevilacqua et al. (2022). The dataset is split into $10k-1k-1k$ for training, validating, and testing. The validation set is used for hyper-parameter search, with learning rate searched in $\{0.001, 0.005, 0.05\}$, and batch size searched in $\{64, 128\}$.

The MAE scores from test data are reported in table 6. Our models rank the $3rd$ among all the models. All the baseline results are from recent years. PPGN results are from Martinkus et al. (2023), and the others are from Bevilacqua et al. (2022).

Table 6: Regression on ZINC12 dataset. MAE is reported.

Method	ZINC (MAE)
PNA (Corso et al., 2020)	0.188 ± 0.004
DGN (Beaini et al., 2021)	0.168 ± 0.003
SMP (Vignac et al., 2020)	0.138
GIN (Xu et al., 2019)	0.252 ± 0.017
HIMP (Fey et al., 2020)	0.151 ± 0.006
GSN (Bouritsas et al., 2023)	0.140 ± 0.006
CIN-SMALL (Bodnar et al., 2021a)	0.094 ± 0.004
DSS-GNN (GIN) (EGO+) (Bevilacqua et al., 2022)	0.102 ± 0.003
PPGN (Maron et al., 2019a)	0.256 ± 0.054
Order-2 GNN-b	0.146 ± 0.007
Order-3 GNN-a	0.110 ± 0.005
Rank	3^{rd}

7. Conclusion

This research contributes to graph neural network models with high expressive power and low complexity. We introduced a compact set of orthogonal bases for linear layers as the basic building blocks of GNNs and proved that the GNNs are permutation invariant. We further compared the expressive power of these networks with the k-WL algorithm in graph isomorphism testing. Specifically, we have a GNN model that uses order-k input tensors and achieves k-WL expressiveness, and another GNN model achieves k-FWL expressiveness. Their expressive power is theoretically proven and verified by real graph isomorphism tests. The proposed GNN models also demonstrated superior discriminative power compared to known graph neural network models when applied to graph classification on benchmark datasets. While their ability to perform other tasks, such as counting graph substructures, is unknown at this point, it could be an interesting topic for future work.

Acknowledgments

This research was supported in part by National Science Foundation (NSF) awards DMS-2027725 and DMS-2415227.

Appendix A. Appendix

A.1 Equivalence classes for $L : \mathbb{R}^{n^k} \to \mathbb{R}^{n^l}$ with $k = 2$

When $k = 2$, $\dim(\Gamma_k) = 3(2^k - 1) - k = 7$, there are a total of 7 equivalence classes for output orders $l = 0, 1, 2$.

- $L : \mathbb{R}^{n^2} \to \mathbb{R}^{n^2}$. For the 4-tuple index $\mathcal{I} = \{i_1, i_2, i_3, i_4\}$, there are 2 equivalence classes $\gamma^{(2)} = \{\gamma_1^{(2)}, \gamma_2^{(2)}\}$:

$$\gamma_1^{(2)} = \{\{1,3\},\{2\},\{4\}\}$$
$$\gamma_2^{(2)} = \{\{2,4\},\{1\},\{3\}\}$$

- $L : \mathbb{R}^{n^2} \to \mathbb{R}^{n^1}$. For the 3-tuple index $\mathcal{I} = \{i_1, i_2, i_3\}$, there are 3 equivalence classes $\gamma^{(1)} = \{\gamma_1^{(1)}, \gamma_2^{(1)}, \gamma_3^{(1)}\}$:

$$\gamma_1^{(1)} = \{\{1,2\},\{3\}\}$$
$$\gamma_2^{(1)} = \{\{1,3\},\{2\}\}$$
$$\gamma_3^{(1)} = \{\{1,2,3\}\}$$

- $L : \mathbb{R}^{n^2} \to \mathbb{R}^{n^0}$. For the 2-tuple index $\mathcal{I} = \{i_1, i_2\}$, there are 2 equivalence classes $\gamma^{(0)} = \{\gamma_1^{(0)}, \gamma_2^{(0)}\}$:

$$\gamma_1^{(0)} = \{\{1\},\{2\}\}$$
$$\gamma_2^{(0)} = \{\{1,2\}\}$$

A.2 Equivalence classes for $L : \mathbb{R}^{n^k} \to \mathbb{R}^{n^l}$ with $k = 3$

With $k = 3$, the dimension of the bases for all output orders l is bounded by $\dim(\Gamma_k) = 3(2^k - 1) - k = 18$, there are a total of 18 equivalence classes.

- $L : \mathbb{R}^{n^3} \to \mathbb{R}^{n^3}$. For the 6-tuple index $\mathcal{I} = \{i_1, i_2, i_3, i_4, i_5, i_6\}$, there are 3 equivalence classes $\gamma^{(3)} = \{\gamma_1^{(3)}, \gamma_2^{(3)}, \gamma_3^{(3)}\}$:

$$\gamma_1^{(3)} = \{\{2,5\},\{3,6\},\{1\},\{4\}\}$$
$$\gamma_2^{(3)} = \{\{1,4\},\{3,6\},\{2\},\{5\}\}$$
$$\gamma_3^{(3)} = \{\{1,4\},\{2,5\},\{3\},\{6\}\}$$

- $L : \mathbb{R}^{n^3} \to \mathbb{R}^{n^2}$. For the 5-tuple index $\mathcal{I} = \{i_1, i_2, i_3, i_4, i_5\}$, there are 6 equivalence classes $\gamma^{(2)} = \{\gamma_1^{(2)}, \gamma_2^{(2)}, \gamma_3^{(2)}, \gamma_4^{(2)}, \gamma_5^{(2)}, \gamma_6^{(2)}\}$:

$$\gamma_1^{(2)} = \{\{1,4\},\{2,5\},\{3\}\}$$
$$\gamma_2^{(2)} = \{\{1,3\},\{2,5\},\{4\}\}$$
$$\gamma_3^{(2)} = \{\{1,3\},\{2,4\},\{5\}\}$$
$$\gamma_4^{(2)} = \{\{1,3,4\},\{2,5\}\}$$
$$\gamma_5^{(2)} = \{\{2,4,5\},\{1,3\}\}$$
$$\gamma_6^{(2)} = \{\{2,3,5\},\{1,4\}\}$$

- $L : \mathbb{R}^{n^3} \to \mathbb{R}^{n^1}$. For the 4-tuple index $\mathcal{I} = \{i_1, i_2, i_3, i_4\}$, there are 7 equivalence classes $\gamma^{(1)} = \{\gamma_1^{(1)}, \gamma_2^{(1)}, \gamma_3^{(1)}, \gamma_4^{(1)}, \gamma_5^{(1)}, \gamma_6^{(1)}, \gamma_7^{(1)}\}$:

$$\begin{aligned}
\gamma_1^{(1)} &= \{\{1,2\},\{3\},\{4\}\} \\
\gamma_2^{(1)} &= \{\{1,3\},\{2\},\{4\}\} \\
\gamma_3^{(1)} &= \{\{1,4\},\{2\},\{3\}\} \\
\gamma_4^{(1)} &= \{\{1,4\},\{2,3\}\} \\
\gamma_5^{(1)} &= \{\{1,2\},\{3,4\}\} \\
\gamma_6^{(1)} &= \{\{1,3\},\{2,4\}\} \\
\gamma_7^{(1)} &= \{\{1,2,3,4\}\}
\end{aligned}$$

- $L : \mathbb{R}^{n^3} \to \mathbb{R}^{n^0}$. For the 3-tuple index $\mathcal{I} = \{i_1, i_2, i_3\}$, there are 2 equivalence classes $\gamma^{(0)} = \{\gamma_1^{(0)}, \gamma_2^{(0)}\}$:

$$\begin{aligned}
\gamma_1^{(0)} &= \{\{1\},\{2\},\{3\}\} \\
\gamma_2^{(0)} &= \{\{1,2,3\}\}
\end{aligned}$$

A.3 Equivalence classes for $L : \mathbb{R}^{n^k} \to \mathbb{R}^{n^l}$ with $k = 4$

With $k = 4$, the dimension of the bases for all output orders is bounded by $\dim(\Gamma_k) = 3(2^k - 1) - k = 41$, so there are a total of 41 equivalence classes.

- $L : \mathbb{R}^{n^4} \to \mathbb{R}^{n^4}$. For the 8-tuple index $\mathcal{I} = \{i_1, i_2, i_3, i_4, i_5, i_6, i_7, i_8\}$, there are 4 equivalence classes $\gamma^{(4)} = \{\gamma_1^{(4)}, \gamma_2^{(4)}, \gamma_3^{(4)}, \gamma_4^{(4)}\}$:

$$\begin{aligned}
\gamma_1^{(4)} &= \{\{1,5\},\{2,6\},\{3,7\},\{4\},\{8\}\} \\
\gamma_2^{(4)} &= \{\{1,5\},\{2,6\},\{4,8\},\{3\},\{7\}\} \\
\gamma_3^{(4)} &= \{\{1,5\},\{3,7\},\{4,8\},\{2\},\{6\}\} \\
\gamma_4^{(4)} &= \{\{2,6\},\{3,7\},\{4,8\},\{1\},\{5\}\}
\end{aligned}$$

- $L : \mathbb{R}^{n^4} \to \mathbb{R}^{n^3}$. For the 7-tuple index $\mathcal{I} = \{i_1, i_2, i_3, i_4, i_5, i_6, i_7\}$, there are 10 equivalence classes $\gamma^{(3)} = \{\gamma_1^{(3)}, \ldots, \gamma_{10}^{(3)}\}$:

$$\begin{aligned}
\gamma_1^{(3)} &= \{\{1,4\},\{2,5\},\{3,6\},\{7\}\} \\
\gamma_2^{(3)} &= \{\{1,4\},\{2,5\},\{3,7\},\{6\}\} \\
\gamma_3^{(3)} &= \{\{1,4\},\{2,6\},\{3,7\},\{5\}\} \\
\gamma_4^{(3)} &= \{\{1,5\},\{2,6\},\{3,7\},\{4\}\} \\
\gamma_5^{(3)} &= \{\{1,4,5\},\{2,6\},\{3,7\}\} \\
\gamma_6^{(3)} &= \{\{1,4,6\},\{2,5\},\{3,7\}\} \\
\gamma_7^{(3)} &= \{\{1,4,7\},\{2,5\},\{3,6\}\} \\
\gamma_8^{(3)} &= \{\{1,4\},\{2,5,6\},\{3,7\}\} \\
\gamma_9^{(3)} &= \{\{1,4\},\{2,5,7\},\{3,6\}\} \\
\gamma_{10}^{(3)} &= \{\{1,4\},\{2,5\},\{3,6,7\}\}
\end{aligned}$$

- $L : \mathbb{R}^{n^4} \to \mathbb{R}^{n^2}$. For the 6-tuple index $\mathcal{I} = \{i_1, i_2, i_3, i_4, i_5, i_6\}$, there are 16 equivalence classes $\gamma^{(2)} = \{\gamma_1^{(2)}, \ldots, \gamma_{16}^{(2)}\}$:

$$\begin{aligned}
\gamma_1^{(2)} &= \{\{1,5\},\{2,6\},\{3\},\{4\}\} \\
\gamma_2^{(2)} &= \{\{1,4\},\{2,6\},\{3\},\{5\}\} \\
\gamma_3^{(2)} &= \{\{1,4\},\{2,5\},\{3\},\{6\}\} \\
\gamma_4^{(2)} &= \{\{1,3\},\{2,6\},\{4\},\{5\}\} \\
\gamma_5^{(2)} &= \{\{1,3\},\{2,5\},\{4\},\{6\}\} \\
\gamma_6^{(2)} &= \{\{1,3\},\{2,4\},\{5\},\{6\}\} \\
\gamma_7^{(2)} &= \{\{1,3,4,5\},\{2,6\}\} \\
\gamma_8^{(2)} &= \{\{1,3,4,6\},\{2,5\} \\
\gamma_9^{(2)} &= \{\{1,3,5,6\},\{2,4\}\} \\
\gamma_{10}^{(2)} &= \{\{2,4,5,6\},\{1,3\}\} \\
\gamma_{11}^{(2)} &= \{\{1,5\},\{2,6\},\{3,4\}\} \\
\gamma_{12}^{(2)} &= \{\{1,4\},\{2,6\},\{3,5\}\} \\
\gamma_{13}^{(2)} &= \{\{1,4\},\{2,5\},\{3,6\}\} \\
\gamma_{14}^{(2)} &= \{\{1,3\},\{2,6\},\{4,5\}\} \\
\gamma_{15}^{(2)} &= \{\{1,3\},\{2,5\},\{4,6\}\} \\
\gamma_{16}^{(2)} &= \{\{1,3\},\{2,4\},\{5,6\}\}
\end{aligned}$$

- $L : \mathbb{R}^{n^4} \to \mathbb{R}^{n^1}$. For the 5-tuple index $\mathcal{I} = \{i_1, i_2, i_3, i_4, i_5\}$, there are 9 equivalence classes $\gamma^{(1)} = \{\gamma_1^{(1)}, \ldots, \gamma_9^{(1)}\}$:

$$\begin{aligned}
\gamma_1^{(1)} &= \{\{1,2\},\{3\},\{4\},\{5\}\} \\
\gamma_2^{(1)} &= \{\{1,3\},\{2\},\{4\},\{5\}\} \\
\gamma_3^{(1)} &= \{\{1,4\},\{2\},\{3\},\{5\}\} \\
\gamma_4^{(1)} &= \{\{1,5\},\{2\},\{3\},\{4\}\} \\
\gamma_5^{(1)} &= \{\{1,2,3,4,5\}\} \\
\gamma_6^{(1)} &= \{\{1,5\},\{2,3,4\}\} \\
\gamma_7^{(1)} &= \{\{1,4\},\{2,3,5\}\} \\
\gamma_8^{(1)} &= \{\{1,3\},\{2,4,5\}\} \\
\gamma_9^{(1)} &= \{\{1,2\},\{3,4,5\}\}
\end{aligned}$$

- $L : \mathbb{R}^{n^4} \to \mathbb{R}^{n^0}$. For the 4-tuple index $\mathcal{I} = \{i_1, i_2, i_3, i_4\}$, there are 2 equivalence classes $\gamma^{(0)} = \{\gamma_1^{(0)}, \gamma_2^{(0)}\}$:

$$\begin{aligned}
\gamma_1^{(0)} &= \{\{1\},\{2\},\{3\},\{4\}\} \\
\gamma_2^{(0)} &= \{\{1,2,3,4\}\}
\end{aligned}$$

A.4 Proof of Theorem 2

Proof. To show that L is permutation invariant (for $l = 0$) or equivariant (for $l \geq 1$), it suffices to show that (1) $B^{\gamma^{(l)}} X$ is invariant or equivariant, and (2) $C^{\beta^{(l)}}$ is invariant or equivariant. It suffices to show the conclusion holds for any specific value of $l \in \{0, \ldots, k\}$. The superscript in $\gamma^{(l)}$ and $\beta^{(l)}$ denotes an equivalence class for the output order l.

Suppose the permutation operator is g. We know that the index for B^γ is an m-tuple $(i_1, \ldots, i_m)$ with $m = k + l$, and $g(i_1, \ldots, i_m) = (g(i_1), \ldots, g(i_m))$. If $(i_1, \ldots, i_m) \in \gamma$ for some γ defined in the equivalence classes, then $(g(i_1), \ldots, g(i_m)) \in \gamma$. This is due to the facts that the equivalence class γ is defined based on the equality patterns of $(i_1, \ldots, i_m)$ and g is a bijective function (i.e., $g(a) = g(b)$ if and only if $a = b$). Therefore, the permutation operation does not change the equivalence class of an m-tuple index.

We exemplify it by using an example: Let $(i_1, \ldots, i_6) \in \gamma$, in which $i_2 = i_5$, $i_3 = i_6$, and $i_1 \neq i_2 \neq i_3 \neq i_4$. Since g is a bijection, we have $g(i_2) = g(i_5)$, $g(i_3) = g(i_6)$, and $g(i_1) \neq g(i_2) \neq g(i_3) \neq g(i_4)$, then we have $(g(i_1), \ldots, g(i_6)) \in \gamma$.

We can write $(i_1, \ldots, i_m)$ as $(i_1, \ldots, i_l, i_{l+1}, \ldots, i_{l+k})$, with $(i_{l+1}, \ldots, i_{l+k})$ correspond to an entry in the input tensor X, and $(i_1, \ldots, i_l)$ correspond to an entry in the output tensor. The permutation action moves an entry in X from position $(i_{l+1}, \ldots, i_{l+k})$ to $g(i_{l+1}, \ldots, i_{l+k}) = (g(i_{l+1}), \ldots, g(i_{l+k}))$, therefore $X_{i_{l+1},\ldots,i_{l+k}} = (g \cdot X)_{g(i_{l+1}),\ldots,g(i_{l+k})}$, and it is still multiplied by the same value of 1 in B^γ if $(i_1, \ldots, i_m) \in \gamma$ (or 0 if $(i_1, \ldots, i_m) \notin \gamma$).

For $l = 0$, $B^\gamma(g \cdot X)$ gives the same result as $B^\gamma X$, which is a scalar; for $l \geq 1$, the l-tuple index for the output tensor $(g(i_1), \ldots, g(i_l)) = g(i_1, \ldots, i_l)$ serves to permute the

output of $B^\gamma X$ by g. Therefore, we have

$$B^\gamma(g \cdot X) = B^\gamma X, \text{ for } l = 0;$$

$$B^\gamma(g \cdot X) = g \cdot (B^\gamma X), \text{ for } l \geq 1.$$

The same argument also applies to the bias term tensor C^β, which is considered as a special case by setting $k = 0$ in $m = k + l$. □

A.5 Preliminary on the k-WL and k-FWL graph-isomorphism tests

A.5.1 k-WL

The classical Weisfeiler-Leman algorithm is a graph-isomorphism test based on the color refinement algorithm in Weisfeiler and Leman (1968). Given a graph $G = (V, E)$, the algorithm outputs its color distribution upon termination. If two graphs have the same color distribution, then the two graphs are considered isomorphic.

The k-WL algorithm is a generalization of the graph-isomorphism tests by defining a node as a k-tuple, i.e., $v = (i_1, \ldots, i_k)$, with $(i_1, \ldots, i_k) \in V^k$ (see Grohe and Otto, 2015; Grohe, 2017). Retrospectively, the classical WL is called 1-WL because each node is represented by a 1-tuple, which is just the node id.

We refer to the singleton $v = (i_1)$ as a vertex, and refer to the k-tuple $v = (i_1, \ldots, i_k)$ as a node. The classical WL keeps refining the color of one vertex by aggregating the colors of its neighbors until there are no more updates; the k-WL updates the color of a k-tuple node by using a similar manner, except that the neighborhood definition is different.

$N_j(v)$, the j-th neighborhood of node v, for $j \in [k]$, is given as

$$N_j(v) = \{(i_1, \ldots, i_{j-1}, w, i_{j+1}, \ldots, i_k) \mid w \in [n]\}.$$

In k-WL tests, the initial color assignment to a node is based on the node's atomic type (Grohe, 2017; Morris et al., 2019). During each iteration, the k-WL algorithm collects the neighborhood color information and then concatenate the colors of the j-th neighborhood, for $j \in [k]$, to a node's own color, and then a hash function is applied to update the node's color. With $k = 2$, the input to the hash function takes three arguments,

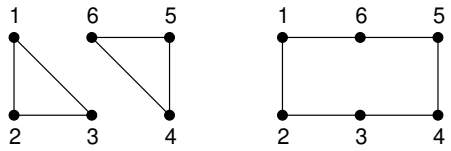
$$c_v^l = \text{hash}\left(c_v^{l-1}, \{c_u^{l-1} | u \in N_1(v)\}, \{c_u^{l-1} | u \in N_2(v)\}\right)$$

The first argument c_v^{l-1} is node v's own color in previous iteration, the second argument is the colors of nodes in $N_1(v)$, and the third argument is the colors of nodes in $N_2(v)$. The colors of the j-th neighborhood is a multiset.

Next, we walk through an example using the two graphs in Test 1 (see Figure 4). Consider 2-WL test. Each node $v = (i_1, i_2)$ is a 2-tuple, so we use a matrix to visualize the color map of the entire graph, with i_1 being the row index, and i_2 being the column index.

Initial color assignment. In graph H, initially there are two atomic types for all nodes, {connected, disconnected}. We color the nodes in red (type 1, connected) and yellow (type2, disconnected). See Figure 5(a) for initial color map. Graph G also has only two atomic types, and is colored as in Figure 6(a).

k-WL neighborhood. Consider a node $v = (i_1, i_2)$. The first neighborhood $N_1(v) = \{(w, i_2) | w \in [n]\}$, and the second neighborhood $N_2(v) = \{(i_1, w) | w \in [n]\}$. In the matrix



(a) H (b) G

Figure 4: Graphs H and G in Test 1 with node ids.

(a) $H^{(0)}$ (b) $v = (2, 3)$ (c) $v = (2, 4)$ (d) $H^{(1)}$

Figure 5: k-WL test for graph H. (a) initial color assignment, (b-c) k-WL neighborhood for two nodes, and (d) an update step.

(a) $G^{(0)}$ (b) $v = (2, 3)$ (c) $v = (2, 4)$ (d) $G^{(1)}$

Figure 6: k-WL test for graph G. (a) initial color assignment, (b-c) k-WL neighborhood for two nodes, and (d) an update step.

representation, the first neighborhood $N_1(v)$ includes nodes in the same column as v, and the second neighborhood $N_2(v)$ includes nodes in the same row as v.

Consider a node $v = (2, 3)$ in graph H, the colors in the first and second neighborhoods are shown in Figure 5(b). The colors in $N_1(v)$ are 3R+3Y (3 red, 3 yellow), and the colors in $N_2(v)$ are 3R+3Y. Note that 3R+3Y=3Y+3R since a multiset is an unordered structure.

Color update. Across the entire graph H, there are only two atomic types: (R, 3R+3Y, 3R+3Y) as shown in node (2,3), and (Y, 3R+3Y, 3R+3Y) as shown in node (2,4). The hash function is an injective function. If there are only two distinct input patterns to the hash function, there will be only two distinct colors it outputs. We again color type 1 in red and type 2 in yellow.

$$\text{R} \leftarrow \text{hash(R, 3R+3Y, 3R+3Y) and Y} \leftarrow \text{hash(Y, 3R+3Y, 3R+3Y)}$$

Figures 5(d) and 6(d) show the results after one update step for graph H and G, respectively.

After one update step, the color pattern of the matrix remains the same for both G and H. The algorithm terminates. The 2-WL algorithm outputs the colors of all nodes in the

matrix, which is 18R+18Y. Since graph G and H both have the same color distribution, they are regarded as the same. Thus, 2-WL fails to distinguish the two graphs.

A.5.2 k-FWL

The test procedure studied in Cai et al. (1992); Morris et al. (2019) is referred to as k-Folklore-WL (k-FWL for short), which is a variant of k-WL. As in k-WL, a node is also defined as a k-tuple, and the procedure also iterates to update the colors of nodes until there are no more updates. However, the k-FWL neighborhood definition is different from the k-WL neighborhood. The k-FWL neighborhood is a multiset $\{N_j^F(v)|j \in [n]\}$. With $k = 2$, each element $N_j^F(v) = ((j, i_2), (i_1, j))$ is an ordered 2-tuple.

The input to the hash function is also different from the k-WL algorithm. The hash function takes two arguments regardless the value of k: the node's own color and the neighborhood's color.

$$c_v^l = \text{hash}\left(c_v^{l-1}, \left\{c_u^{l-1} \mid u \in N_j^F(v), j \in [n]\right\}\right)$$

Consider graph H. The initial color assignment is the same as in k-WL. The color of the neighborhood for the node $v = (2, 3)$ is 3RR+3YY (see Figure 7(b)), while for node $v = (2, 4)$ is 3YR+3RY (see Figure 7(c)). Across the graph H, there are two atomic types: (R, 3RR+3YY) as in node $v = (2, 3)$ and (Y, 3YR+3RY) as in node $v = (2, 4)$. We color type 1 in red and type 2 in yellow again.

$$\text{R} \leftarrow \text{hash(R, 3RR+3YY) and Y} \leftarrow \text{hash(Y, 3YR+3RY)}$$

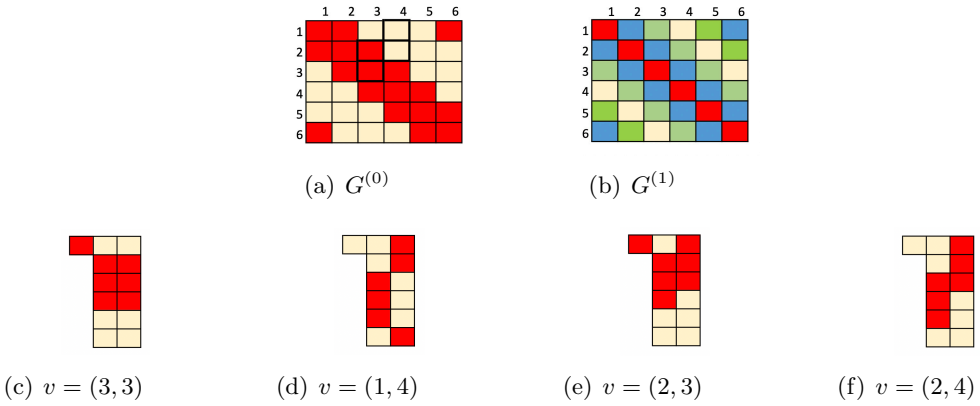
After one updating step, the color pattern of the entire graph remains the same. So the algorithm terminates, and outputs a color distribution of 18R+18Y.

(a) $H^{(0)}$ (b) $v = (2, 3)$ (c) $v = (2, 4)$ (d) $H^{(1)}$

Figure 7: k-FWL test for graph H. (a) initial color assignment, (b-c) k-FWL neighborhood for two nodes, and (d) an update step.

Now consider graph G. The initial color assignment is the same as in k-WL. However, after the initial color assignment, there are four atomic types in graph G, represented by nodes (3,3), (1,4), (2,3) and (2,4) (see Figure 8(c-f)). We add colors blue and green for type 3 and type 4.

- $v = (3, 3)$, R $\leftarrow$ hash(R, 3RR+3YY)
- $v = (1, 4)$, Y $\leftarrow$ hash(Y, 3RY+3YR)
- $v = (2, 3)$, B $\leftarrow$ hash(R, 2RR+2YY+1RY+1YR)



(a) $G^{(0)}$ (b) $G^{(1)}$

(c) $v = (3, 3)$ (d) $v = (1, 4)$ (e) $v = (2, 3)$ (f) $v = (2, 4)$

Figure 8: k-FWL test for graph G. (a) initial color assignment, (c-f) k-FWL neighborhood for four nodes, and (b) an update step.

- $v = (2, 4)$, G $\leftarrow$ hash(Y, 1RR+1YY+2RY+2YR)

Figure 8(b) shows the result after one updating step for graph G. The algorithm does not terminate after one updating step. It will continue to update. It is easy to see that there are already more than four atomic types at this point. 2-FWL can distinguish the two graphs H and G since they have different color distributions.

References

Waiss Azizian and Marc Lelarge. Expressive power of invariant and equivariant graph neural networks. In *International Conference on Learning Representations*, 2021.

Peter W Battaglia, Razvan Pascanu, Matthew Lai, Danilo Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics. In *Advances in Neural Information Processing Systems*, 2016.

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

Beatrice Bevilacqua, Fabrizio Frasca, Derek Lim, Balasubramaniam Srinivasan, Chen Cai, Gopinath Balamurugan, Michael M Bronstein, and Haggai Maron. Equivariant subgraph aggregation networks. In *International Conference on Learning Representations*, 2022.

Cristian Bodnar, Fabrizio Frasca, Nina Otter, Yu Guang Wang, Pietro Liò, Guido Montúfar, and Michael Bronstein. Weisfeiler and Lehman go cellular: CW networks. In *Advances in Neural Information Processing Systems*, pages 2625–2640, 2021a.

Cristian Bodnar, Fabrizio Frasca, Yuguang Wang, Nina Otter, Guido Montufar, Pietro Liò, and Michael Bronstein. Weisfeiler and Lehman go topological: Message passing simplicial networks. In *ICLR Workshop on Geometrical and Topological Representation Learning*, 2021b.

Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M. Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(1):657–668, 2023.

Jin-Yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992.

R. Qi Charles, Hao Su, Mo Kaichun, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3D classification and segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, 2017.

Ting Chen, Song Bian, and Yizhou Sun. Are powerful graph neural nets necessary? a dissection on graph classification. *arXiv:1905.04579*, 2019a.

Zhengdao Chen, Soledad Villar, Lei Chen, and Joan Bruna. On the equivalence between graph isomorphism testing and function approximation with gnns. In *Advances in Neural Information Processing Systems*, 2019b.

Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. Can graph neural networks count substructures? In *Advances in Neural Information Processing Systems*, pages 10383–10395, 2020.

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

David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems*, 2015.

Vijay Prakash Dwivedi, Chaitanya K Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *Journal of Machine Learning Research*, 24(43):1–48, 2023.

Matthias Fey, Jan-Gin Yuen, and Frank Weichert. Hierarchical inter-message passing for learning on molecular graphs. In *ICML Graph Representation Learning and Beyond (GRL+) Workshop*, 2020.

Hongyang Gao and Shuiwang Ji. Graph representation learning via hard and channel-wise attention networks. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 741–749, 2019.

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*, pages 1263–1272, 2017.

Rafael Gómez-Bombarelli, Jennifer N. Wei, David Duvenaud, José Miguel Hernández-Lobato, BenjamíÂn Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre,

Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2): 268–276, 2018.

Martin Grohe. *Descriptive complexity, canonisation, and definable graph structure theory*, volume 47. Cambridge University Press, 2017.

Martin Grohe and Martin Otto. Pebble games and linear equations. *The Journal of Symbolic Logic*, 80(3):797–844, 2015.

Jason Hartford, Devon Graham, Kevin Leyton-Brown, and Siamak Ravanbakhsh. Deep models of interactions across sets. In *International Conference on Machine Learning*, pages 1909–1918, 2018.

Ningyuan Teresa Huang and Soledad Villar. A short tutorial on the Weisfeiler-Lehman test and its variants. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8533–8537. IEEE, 2021.

Chaitanya K. Joshi, Cristian Bodnar, Simon V Mathis, Taco Cohen, and Pietro Lio. On the expressive power of geometric graph neural networks. In *International Conference on Machine Learning*, 2023.

Nicolas Keriven and Gabriel Peyré. Universal invariant and equivariant graph neural networks. In *Advances in Neural Information Processing Systems*, volume 32, pages 7092–7101, 2019.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

Soheil Kolouri, Navid Naderializadeh, Gustavo K Rohde, and Heiko Hoffmann. Wasserstein embedding for graph learning. In *International Conference on Learning Representations*, 2021.

Risi Kondor, Nino Shervashidze, and Karsten M Borgwardt. The graphlet spectrum. In *International Conference on Machine Learning*, pages 529–536, 2009.

Risi Kondor, Hy Truong Son, Horace Pan, Brandon Anderson, and Shubhendu Trivedi. Covariant compositional networks for learning graphs. In *International Conference on Learning Representations, Workshop Track*, 2018.

Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. Distance encoding–design provably more powerful gnns for structural representation learning. In *Advances in Neural Information Processing Systems*, 2020.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. In *International Conference on Learning Representations*, 2016.

Xin Liu, Haojie Pan, Mutian He, Yangqiu Song, Xin Jiang, and Lifeng Shang. Neural subgraph isomorphism counting. In *SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1959–1969, 2020.

Marcelo Daniel Gutierrez Mallea, Peter Meltzer, and Peter J Bentley. Capsule neural networks for graph classification using explicit tensorial graph representations. *arXiv:1902.08399*, 2019.

Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. In *Advances in Neural Information Processing Systems*, 2019a.

Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. In *International Conference on Learning Representations*, 2019b.

Haggai Maron, Ethan Fetaya, Nimrod Segol, and Yaron Lipman. On the universality of invariant networks. In *International Conference on Machine Learning*, pages 4363–4371, 2019c.

Karolis Martinkus, Pál András Papp, Benedikt Schesch, and Roger Wattenhofer. Agent-based graph neural networks. In *International Conference on Learning Representations*, 2023.

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 *AAAI Conference on Artificial Intelligence*, pages 4602–4609, 2019.

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

Marion Neumann, Roman Garnett, Christian Bauckhage, and Kristian Kersting. Propagation kernels: efficient graph kernels from propagated information. *Machine Learning*, 102 (2):209–245, 2016.

Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International Conference on Machine Learning*, pages 2014–2023. PMLR, 2016.

Pál András Papp, Karolis Martinkus, Lukas Faber, and Roger Wattenhofer. Dropgnn: random dropouts increase the expressiveness of graph neural networks. In *Advances in Neural Information Processing Systems*, 2021.

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

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, 2009.

Kristof T Schütt, Farhad Arbabzadah, Stefan Chmiela, Klaus R Müller, and Alexandre Tkatchenko. Quantum-chemical insights from deep tensor neural networks. *Nature Communications*, 8(1):1–8, 2017.

Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-Lehman graph kernels. *Journal of Machine Learning Research*, 12(9):2539–2561, 2011.

Teague Sterling and John J. Irwin. ZINC 15 — ligand discovery for everyone. *Journal of Chemical Information and Modeling*, 55(11):2324–2337, 2015.

Fan-Yun Sun, Jordan Hoffman, Vikas Verma, and Jian Tang. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. In *International Conference on Learning Representations*, 2019.

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

Clement Vignac, Andreas Loukas, and Pascal Frossard. Building powerful and equivariant graph neural networks with structural message-passing. In *Advances in Neural Information Processing Systems*, pages 14143–14155, 2020.

B. Weisfeiler and A.A. Leman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 2(9):12–16, 1968.

Boris Weisfeiler. *On Construction and Identification of Graphs*. Springer, 1976.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1365–1374, 2015.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in Neural Information Processing Systems*, 2017.

Bohang Zhang, Lingxiao Zhao, and Haggai Maron. On the expressive power of spectral invariant graph neural networks. In *International Conference on Machine Learning*, 2024.

Muhan Zhang and Pan Li. Nested graph neural networks. In *Advances in Neural Information Processing Systems*, 2021.

Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *AAAI Conference on Artificial Intelligence*, pages 4438–4445, 2018.