From Graph Low-Rank Global Attention to 2-FWL Approximation

Omri Puny[1] Heli Ben-Hamu[1] Yaron Lipman[1]

Abstract

Graph Neural Networks are known to have an expressive power bounded by that of the vertex coloring algorithm (Xu et al., 2019a; Morris et al., 2018). However, for rich node features, such a bound does not exist and GNNs can be shown to be universal. Unfortunately, expressive power alone does not imply good generalization.

We suggest the Low-Rank Global Attention (LRGA) module, taking advantage of the efficiency of low rank matrix-vector multiplication, that improves the algorithmic alignment (Xu et al., 2019b) of GNNs with the more powerful 2-folklore Weisfeiler-Lehman (FWL) algorithm. Furthermore, we provide a sample complexity bound for the module using kernel feature map interpretation of 2-FWL. Empirically, augmenting existing GNN layers with LRGA produces state of the art results on most datasets in a GNN standard benchmark.

1. Introduction

Perhaps the most commonly used family of GNNs are message-passing neural networks (Gilmer et al., 2017), built by aggregating messages from local neighborhoods at each layer. Many GNN variants have been shown to be an instance of this family (Duvenaud et al., 2015; Li et al., 2016; Battaglia et al., 2016; Niepert et al., 2016; Hamilton et al., 2017; Monti et al., 2017; Veličković et al., 2018; Bresson and Laurent, 2017; Xu et al., 2019a; Bruna et al., 2014; Defferrard et al., 2016; Kipf and Welling, 2016; Maron et al., 2019b). In a recent analysis of the expressive power of such models, (Xu et al., 2019a; Morris et al., 2018) have shown that message-passing neural networks are at most as powerful as the vertex coloring algorithm also known as the 1-Weisfeiler-Lehman (WL) test. 1-WL is part of the k-WL hierarchy of increasing power and complexity iterative algorithms aimed at solving graph isomorphism. This analysis led to the design of new architectures (Morris et al., 2018; Maron et al., 2019a) mimicking higher orders of the k-WL family.

Although expressive power bounds on GNNs exist, empirically in many datasets, GNNs are able to fit the train data well. Thus indicating the expressive power of these models might not be the main roadblock to a successful generalization. Therefore, we focus our efforts on strengthening GNNs from a *generalization* point of view. Towards that goal we propose the Low-rank global attention (LRGA) module which can be augmented to any GNN layer. We define a κ-rank attention matrix, where κ is a parameter, that requires $O(\kappa|V|)$ memory and can be applied in $O(\kappa^2|V|)$ computational complexity, in contrast to standard attention modules that apply $|V| \times |V|$ attention matrix to node data with $O(|V|^3)$ computational complexity.

We restrict our attention to a class of graphs called *rich feature graphs* which have their structural information encoded in the node features. Our theoretical analysis of LRGA under the rich feature graph assumption includes: (i) formulating the 2-FWL algorithm, which is strictly stronger than 1-WL, using polynomial kernels; (ii) showing that LRGA aligns with this formulation of 2-FWL, i.e., LRGA can approximate (for sufficiently high κ) the update step of the 2-FWL algorithm with simple functions; and (iii) bounding the sample complexity of the LRGA module when learning the 2-FWL update rule. Although our bound is exponential in the graph size, it nevertheless implies that LRGA can provably learn the 2-FWL step.

2. Low-rank global attention (LRGA)

We consider a graph $G = (V, E)$ where V is the vertex-set of size n and E is the edge-set. Each vertex carries an input feature vector $\boldsymbol{x}_i \in \mathbb{R}^{d_0}$, where d_0 is the input feature dimension. The input vertices' feature vectors are summarized in a matrix $\boldsymbol{X}^0 = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_n)^T \in \mathbb{R}^{n \times d_0}$; in turn, $\boldsymbol{X}^l \in \mathbb{R}^{n \times d_l}$ represents the output of the l^{th} layer of a neural network. We propose the Low-rank global attention

[1] Weizmann Institute of Science. Correspondence to: Omri Puny <omri.puny@weizmann.ac.il>.

Proceedings of the 37th International Conference on Machine Learning, Vienna, Austria, PMLR 108, 2020. Copyright 2020 by the author(s).

(LRGA) module that is added to any GNN layer:

$$\boldsymbol{X}^{l+1} \leftarrow \left[\boldsymbol{X}^{l}, \mathrm{LRGA}(\boldsymbol{X}^{l}), \mathrm{GNN}(\boldsymbol{X}^{l})\right] \tag{1}$$

the brackets imply concatenation along the feature dimension. The LRGA acting on input feature matrix $\boldsymbol{X} \in \mathbb{R}^{n \times d_{\text{in}}}$:

$$\mathrm{LRGA}(\boldsymbol{X}) = \left[\tfrac{1}{\eta(\boldsymbol{X})} m_1(\boldsymbol{X}) \left(m_2(\boldsymbol{X})^T m_3(\boldsymbol{X})\right),\ m_4(\boldsymbol{X})\right] \tag{2}$$

where $m_1, m_2, m_3, m_4 : \mathbb{R}^{n \times d_{\text{in}}} \rightarrow \mathbb{R}^{n \times \kappa}$ are MLPs operating on the feature dimension, that is $m(\boldsymbol{X}) = [m(\boldsymbol{x}_1), \dots, m(\boldsymbol{x}_n)]^T$, and $\kappa \in \mathbb{N}_0$ is a parameter representing the *rank* of the attention module. Lastly, η is a global normalization factor:

$$\eta(\boldsymbol{X}) = \frac{1}{n}\left(\mathbf{1}^T m_1(\boldsymbol{X})\right)\left(m_2(\boldsymbol{X})^T \mathbf{1}\right), \tag{3}$$

where $\mathbf{1} = (1, 1, \dots, 1)^T \in \mathbb{R}^n$. We can think of $\boldsymbol{A} = \eta(\boldsymbol{X})^{-1} m_1(\boldsymbol{X}) m_2(\boldsymbol{X})^T$ as a κ-rank attention matrix that acts globally on the graph's node features. $\eta(\boldsymbol{X})$ represents the expectation of the row sums in $m_1(\boldsymbol{X}) m_2(\boldsymbol{X})^T$, so $\mathbb{E}(\boldsymbol{A}\mathbf{1}) = 1$.

Table 1. Performance on the obg bendhmark datasets.

Model	ogbl-ppa	ogbl-collab
	Hits@100 ± std	Hits@10 ± std
Matrix Factorization	0.3229 ± 0.0094	0.3805 ± 0.0018
Node2Vec	0.2226 ± 0.0083	0.4281 ± 0.0140
GCN	0.1155 ± 0.0153	0.3329 ± 0.0190
GraphSAGE	0.1063 ± 0.0244	0.3121 ± 0.0620
LRGA + GCN	0.2988 ± 0.0211	0.4363 ± 0.0121
LRGA + GCN (large)	**0.3426 ± 0.016**	**0.4541 ± 0.0091**

3. Theoretical Analysis

The LRGA (equation 2) module has an interesting justification in the context of GNNs. In essence, we restrict our attention to a certain graph class with informative node features, called rich feature graphs and show that it algorithmically aligns (Xu et al., 2019b) with the 2-FWL algorithm.

3.1. Rich features can make GNNs universal

The expressive power of GNNs has been shown to be bounded by that of the vertex coloring algorithm (Xu et al., 2019a; Morris et al., 2019). However, it is clear that using more expressive node features can make the graph isomorphism problem easier. Here, we are interested in evaluating the power of GNNs when the node features are informative. As a model for informative node features we define *rich feature graphs* and prove that for this model GNNs are universal, i.e., have maximal expressive power.

Notation. Let $G = (V, E)$, a graph with $D \in \mathbb{N}$ features per node, i.e., $\boldsymbol{x}_i \in \mathbb{R}^D$. In matrix form $\boldsymbol{X} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_n)^T \in \mathbb{R}^{n \times D}$. We further break $\boldsymbol{X}$ into $2d$ blocks, $\boldsymbol{X} = \left[\boldsymbol{X}^1, \dots, \boldsymbol{X}^{2d}\right]$, where consecutive blocks $\boldsymbol{X}^{2\ell-1}, \boldsymbol{X}^{2\ell}$, $\ell \in [d]$, contain the same number of columns.

Definition 1 (Rich feature graph)**.** *A graph $G = (V, E)$ with node features $\boldsymbol{X}$ is a* rich feature graph *if, for some $d \in \mathbb{N}$, there exists a block structure $\boldsymbol{X} = \left[\boldsymbol{X}^1, \dots, \boldsymbol{X}^{2d}\right]$ so that for all $i, j \in [n]$, the vector $\mathbf{Y}_{i,j} = \left[\langle \boldsymbol{x}_i^1, \boldsymbol{x}_j^2\rangle, \langle \boldsymbol{x}_i^3, \boldsymbol{x}_j^4\rangle, \dots, \langle \boldsymbol{x}_i^{2d-1}, \boldsymbol{x}_j^{2d}\rangle\right] \in \mathbb{R}^d$ represents the isomorphism type of the pair (i, j).*

In matrix notation $\mathbf{Y} = \left[\boldsymbol{X}^1(\boldsymbol{X}^2)^T, \dots, \boldsymbol{X}^{2d-1}(\boldsymbol{X}^{2d})^T\right]$. The isomorphism type of a pair (i, j), which represents either an edge or a node of graph G, summarizes all the information this pair carries in graph G.

Hence, rich feature graphs carry all their information in the node features. In fact, every graph can be represented as a rich feature graph. Let $\boldsymbol{M} \in \{0, 1\}^{n \times n}$ be the adjacency matrix of G, and $c_i \in (0, 1)$, $i \in [n]$, representatives of the node's features. Then, $\boldsymbol{Y} = \left[\boldsymbol{M} + \frac{1}{2}I, \boldsymbol{c}\mathbf{1}^T, \mathbf{1}\boldsymbol{c}^T\right]$ represents the isomorphism types of pairs in G. Using the singular value decomposition (SVD) we can write $\boldsymbol{Y} = \left[\boldsymbol{X}^1(\boldsymbol{X}^2)^T, \dots, \boldsymbol{X}^5(\boldsymbol{X}^6)^T\right]$ and $\boldsymbol{X} = \left[\boldsymbol{X}^1, \dots, \boldsymbol{X}^6\right]$ is a rich feature representation for G. In general the isomorphism type is represented as a tensor $\mathbf{Y} \in \mathbb{R}^{n^2 \times d}$. The following proposition shows that GNN in (Battaglia et al., 2018), with a global attribute block, is universal under the rich feature graph assumption.

Proposition 1. *GNNs can approximate an arbitrary continuous function over the class of rich feature graphs with node features $\boldsymbol{X} \in K$ in some compact set $K \subset \mathbb{R}^{n \times D}$.*

3.2. 2-FWL via a polynomial kernel

We next formulate the 2-FWL algorithm using polynomial kernels. Let $G = (V, E)$ be a colored graph with isomorphism types of pairs of vertices represented via a tensor $\mathbf{Y}^0 \in \mathbb{R}^{n^2 \times d_0}$. $\mathbf{Y}^0$ is the initial coloring of the vertex pairs and is set as the input to the 2-FWL algorithm. $\mathbf{Y}^l \in \mathbb{R}^{n^2 \times d_l}$ denotes the coloring after the l^{th} recoloring step. A recoloring step in the algorithm aggregates information from the multiset of neighborhoods colors for each pair. We represent the multiset of neighborhoods colors of the tuple (i, j) with a matrix $\boldsymbol{Z}^l_{(i,j)} \in \mathbb{R}^{n \times 2d_l}$. That is, any permutation of the rows of $\boldsymbol{Z}^l_{(i,j)}$ represent the same multiset. The rows of $\boldsymbol{Z}^l_{(i,j)}$, which represent the elements in the multiset, are $\boldsymbol{z}_k = \left[\mathbf{Y}^l_{i,k}, \mathbf{Y}^l_{k,j}\right] \in \mathbb{R}^{2d_l}$, $k \in [n]$.

The 2-FWL update step of a pair (i, j) is done by concatenating the previous color and an encoding of the neighborhood:

$$\mathbf{Y}^{l+1}_{i,j} = \left[\mathbf{Y}^l_{i,j}, \mathrm{ENC}\left(\boldsymbol{Z}^l_{(i,j)}\right)\right] \tag{4}$$

$\mathrm{ENC} : \mathbb{R}^{n \times 2d_l} \rightarrow \mathbb{R}^{d_{enc}}$ is the encoding function that is invariant to the row-order of its input and maps different multisets to different target vectors.

Multiset encoding. As shown in (Maron et al., 2019a) a collection of Power-sum Multi-symmetric Polynomials

Table 2. Performance on the benchmark GNN datasets.

Model	CLUSTER		PATTERN		CIFAR10		MNIST		TSP	
	# Param	Acc ± std	# Param	Acc ± std	# Param	Acc ± std	# Param	Acc ± std	# Param	F1 ± std
MLP	104305	20.97 ± 0.01	103629	50.13 ± 0.00	106017	56.78 ± 0.12	105717	95.18 ± 0.18	94394	0.548 ± 0.003
GCN	101655	47.82 ± 4.91	100923	74.36 ± 1.59	101657	54.46 ± 0.10	101365	89.99 ± 0.15	108738	0.627 ± 0.003
GraphSAGE	99139	53.90 ± 4.12	98607	81.25 ± 3.84	102907	66.08 ± 0.24	102691	97.20 ± 0.17	98450	0.663 ± 0.003
GIN	103544	52.54 ± 1.03	100884	98.25 ± 0.38	105654	53.28 ± 3.70	105434	93.96 ± 1.30	118574	0.657 ± 0.001
DiffPool	-	-	-	-	108042	57.99 ± 0.45	106538	95.02 ± 0.42	-	-
GAT	110700	54.12 ± 1.21	109936	90.72 ± 2.04	110704	65.48 ± 0.33	110400	95.62 ± 0.13	109250	0.669 ± 0.001
MoNet	104227	45.95 ± 3.39	103775	97.89 ± 0.89	104229	53.42 ± 0.43	104049	90.36 ± 0.47	94274	0.637 ± 0.01
GatedGCN	104355	54.20 ± 3.58	104003	97.24 ± 1.19	104357	69.37 ± 0.48	104217	97.47 ± 0.13	94946	**0.802 ± 0.001**
LRGA + GatedGCN	93482	**62.11 ± 3.47**	104663	**98.68 ± 0.16**	93485	**70.65 ± 0.18**	93395	**98.20 ± 0.03**	103347	0.798 ± 0.001

(PMPs) can be used to build the multiset encoding function, ENC. Given a multiset $\boldsymbol{Z} = (\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n)^T \in \mathbb{R}^{n \times 2d}$:

$$\mathrm{ENC}(\boldsymbol{Z}) = \left[\sum_{k=1}^{n} \boldsymbol{z}_k^{\boldsymbol{\alpha}} \;\middle|\; \boldsymbol{\alpha} \in \mathbb{N}_0^{2d}, |\boldsymbol{\alpha}| \leq n \right],$$

where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_{2d})$, and $\boldsymbol{z}^{\boldsymbol{\alpha}} = z_1^{\alpha_1} \cdots z_{2d}^{\alpha_{2d}}$.

We focus on computing a single output coordinate $\boldsymbol{\alpha}$ of the ENC function applied to a particular multiset $\boldsymbol{Z}_{(i,j)}$. This can be computed using matrix multiplication (Maron et al., 2019a): Let $\boldsymbol{\alpha} = (\boldsymbol{\beta}, \boldsymbol{\gamma}) \in \mathbb{N}_0^{2d}$, where $\boldsymbol{\beta}, \boldsymbol{\gamma} \in \mathbb{N}_0^{d}$. Then,

$$\mathrm{ENC}_{\boldsymbol{\alpha}}(\boldsymbol{Z}_{(i,j)}) = \sum_{k=1}^{n} \boldsymbol{z}_k^{\boldsymbol{\alpha}} = \sum_{k=1}^{n} \mathbf{Y}_{i,k}^{\boldsymbol{\beta}} \mathbf{Y}_{k,j}^{\boldsymbol{\gamma}} = (\mathbf{Y}^{\boldsymbol{\beta}} \mathbf{Y}^{\boldsymbol{\gamma}})_{i,j}. \quad (5)$$

By $\mathbf{Y}^{\boldsymbol{\beta}}$ we mean that we apply the multi-power $\boldsymbol{\beta}$ to the feature dimension, i.e., $(\mathbf{Y}^{\boldsymbol{\beta}})_{i,j} = \mathbf{Y}_{i,j}^{\boldsymbol{\beta}}$. This implies that computing the multisets encoding amounts to calculating monomials $\mathbf{Y}^{\boldsymbol{\beta}}, \mathbf{Y}^{\boldsymbol{\gamma}}$ and their matrix multiplications $\mathbf{Y}^{\boldsymbol{\beta}} \mathbf{Y}^{\boldsymbol{\gamma}}$. Thus the 2-FWL update rule, equation 4, can be written in the following matrix form, where we denote $\mathbf{Y} = \mathbf{Y}^{l}$:

$$\mathbf{Y}^{l+1} = \left[\mathbf{Y}, \left[\mathbf{Y}^{\boldsymbol{\beta}} \mathbf{Y}^{\boldsymbol{\gamma}} \mid (\boldsymbol{\beta}, \boldsymbol{\gamma}) \in \mathbb{N}_0^{2d}, |\boldsymbol{\beta}| + |\boldsymbol{\gamma}| \leq n \right] \right] \quad (6)$$

2-FWL via polynomial kernels. Let the node feature matrix $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)^T \in \mathbb{R}^{n \times D}$, and $\mathbf{Y}_{i,j} = \left[\langle \boldsymbol{x}_i^1, \boldsymbol{x}_j^2 \rangle, \ldots, \langle \boldsymbol{x}_i^{2d-1}, \boldsymbol{x}_j^{2d} \rangle \right]$ are the colors they define on the vertex pairs. We show it is possible to compute $\mathbf{Y}^{\boldsymbol{\beta}} \mathbf{Y}^{\boldsymbol{\gamma}}$ directly from $\boldsymbol{X}$ using polynomial feature maps. Indeed,

$$\mathbf{Y}_{i,j}^{\boldsymbol{\beta}} = \prod_{\ell=1}^{d} \langle \boldsymbol{x}_i^{2\ell-1}, \boldsymbol{x}_j^{2\ell} \rangle^{\beta_\ell} = \prod_{\ell=1}^{d} \langle \varphi_{\beta_\ell}(\boldsymbol{x}_i^{2\ell-1}), \varphi_{\beta_\ell}(\boldsymbol{x}_j^{2\ell}) \rangle$$
$$= \prod_{\ell=1}^{d} \langle \varphi_{\beta_\ell}(\boldsymbol{x}_i^{\mathrm{odd}}), \varphi_{\beta_\ell}(\boldsymbol{x}_j^{\mathrm{even}}) \rangle = \langle \varphi_{\boldsymbol{\beta}}(\boldsymbol{x}_i^{\mathrm{odd}}), \varphi_{\boldsymbol{\beta}}(\boldsymbol{x}_j^{\mathrm{even}}) \rangle$$

where the second equality is using the feature maps φ_{β_ℓ} of the (homogeneous) polynomial kernels (Vapnik, 1998), $\langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle^{\beta_\ell}$; the third equality is reformulating the feature maps φ_{β_ℓ} on the vectors $\boldsymbol{x}_i^{\mathrm{odd}} = \left[\boldsymbol{x}_i^1, \ldots, \boldsymbol{x}_i^{2d-1} \right]$, and $\boldsymbol{x}_i^{\mathrm{even}} = \left[\boldsymbol{x}_i^2, \ldots, \boldsymbol{x}_i^{2d} \right]$; and the last equality is due to the closure of kernels to multiplication. We denote the final feature map by $\varphi_{\boldsymbol{\beta}}$. Now, let $\psi_{\boldsymbol{\beta}}(\boldsymbol{x}_i) = \varphi_{\boldsymbol{\beta}}(\boldsymbol{x}_i^{\mathrm{odd}})$ and $\phi_{\boldsymbol{\beta}}(\boldsymbol{x}_i) = \varphi_{\boldsymbol{\beta}}(\boldsymbol{x}_i^{\mathrm{even}})$, then we have:

$$\mathbf{Y}^{\boldsymbol{\beta}} = \psi_{\boldsymbol{\beta}}(\boldsymbol{X}) \phi_{\boldsymbol{\beta}}(\boldsymbol{X})^T,$$

where $\psi_{\boldsymbol{\beta}}(\boldsymbol{X})$ is applying $\psi_{\boldsymbol{\beta}}$ to every row of $\boldsymbol{X}$. Therefore, $\mathbf{Y}^{\boldsymbol{\beta}} \mathbf{Y}^{\boldsymbol{\gamma}}$ can be written directly as a function of the node features $\boldsymbol{X}$ using the feature maps $\phi_{\boldsymbol{\beta}}, \psi_{\boldsymbol{\beta}}, \phi_{\boldsymbol{\gamma}}, \psi_{\boldsymbol{\gamma}}$:

$$\mathbf{Y}^{\boldsymbol{\beta}} \mathbf{Y}^{\boldsymbol{\gamma}} = \psi_{\boldsymbol{\beta}}(\boldsymbol{X}) \phi_{\boldsymbol{\beta}}(\boldsymbol{X})^T \psi_{\boldsymbol{\gamma}}(\boldsymbol{X}) \phi_{\boldsymbol{\gamma}}(\boldsymbol{X})^T. \quad (7)$$

Table 3. Results on ZINC dataset.

Model	ZINC		ZINC (large)	
	# Param	MAE ± std	# Param	MAE ± std
MLP	106970	0.681 ± 0.005	2289351	0.7035 ± 0.003
GCN	103077	0.469 ± 0.002	2189531	0.479 ± 0.007
GraphSage	105031	0.410 ± 0.005	2176751	0.439 ± 0.006
GIN	103079	0.408 ± 0.008	2028509	0.382 ± 0.008
DiffPool	110561	0.466 ± 0.006	2291521	0.448 ± 0.005
GAT	102385	0.463 ± 0.002	2080881	0.471 ± 0.005
MoNet	106002	0.407 ± 0.007	2244343	0.372 ± 0.01
GatedGCN	105875	**0.363 ± 0.009**	2134081	0.338 ± 0.003
LRGA + GatedGCN	94457	0.367 ± 0.008	1989730	**0.285 ± 0.01**

3.3. Algorithmic alignment with 2-FWL

Our goal is to show that LRGA algorithmically aligns with 2-FWL over rich feature graphs providing justification to its improved generalization properties. We consider the notion of algorithmic alignment introduced in (Xu et al., 2019b).

First, we show that LRGA (equation 2) can implement a single multi-power $\boldsymbol{\alpha}$ of the 2-FWL update rule in equation 6. The single head 2-FWL update rule is $\mathbf{Y}^{l+1} = \left[\mathbf{Y}, \mathbf{Y}^{\boldsymbol{\beta}} \mathbf{Y}^{\boldsymbol{\gamma}} \right]$. Using equation 7 the rule over the input node features $\boldsymbol{X}$:

$$\boldsymbol{X}^{l+1} = \left[\boldsymbol{X}, \psi_{\boldsymbol{\beta}}(\boldsymbol{X}) \phi_{\boldsymbol{\beta}}(\boldsymbol{X})^T \psi_{\boldsymbol{\gamma}}(\boldsymbol{X}), \phi_{\boldsymbol{\gamma}}(\boldsymbol{X}) \right]$$

It can be readily checked that the updated node features $\mathbf{X}^{l+1}$ indeed define the updated colors with a single head $\mathbf{Y}^{l+1}$. To finish the argument note that this update equation has the same form as the LRGA. Note that the normalization η in equation 2 is a multiplication by a scalar and therefore has no influence on the colors. We showed:

Theorem 1. *LRGA module can simulate a single head 2-FWL update rule under rich feature graph assumption.*

Bound on LRGA sample complexity. We conclude this section by proving that the learnable modules in LRGA, namely the MLPs m_i, can provably learn the feature maps $\phi_{\boldsymbol{\beta}}, \psi_{\boldsymbol{\beta}}$. Let us denote by $\phi : \mathbb{R}^D \to \mathbb{R}^m$ one of these feature maps. All the m outputs of ϕ consist of monomials $\boldsymbol{x}^{\boldsymbol{\delta}}$, where $\boldsymbol{x} \in \mathbb{R}^D$ and $\boldsymbol{\delta} \in \mathbb{N}_0^D$, $|\boldsymbol{\delta}| \le n$, and $m \le N$, where $N = \binom{n+D}{D}$ is the dimension of all D-variate polynomials of degree at-most n. We will consider a single output coordinate of ϕ, namely $\boldsymbol{x}^{\boldsymbol{\delta}}$.

Corollary 6.2 (Arora et al., 2019) provides a sample complexity bound, denoted $\mathcal{C}_{\mathcal{A}}(g, \epsilon, \delta)$, of a polynomial $g : \mathbb{R}^D \to \mathbb{R}$ of the form $g(\boldsymbol{x}) = \sum_j a_j \langle \boldsymbol{\beta}_j, \boldsymbol{x} \rangle^{p_j}$, where $p_j \in \{1, 2, 4, 6, \dots\}$, $a_j \in \mathbb{R}$, $\boldsymbol{\beta}_j \in \mathbb{R}^D$; ϵ, δ are the PAC learning constants, $\mathcal{A}$ is an over-parameterized, randomly initialized two-layer MLP trained with gradient descent.

Let $\mathcal{B} = \left\{ \boldsymbol{\beta} \in \mathbb{N}_0^D \mid |\boldsymbol{\beta}| \le n \right\}$, and note that there are N elements in $\mathcal{B}$. We assume some fixed ordering in $\mathcal{B}$ is prescribed. Define the sample matrix (multivariate Vandemonde) $\boldsymbol{V} \in \mathbb{R}^{N \times N}$ by $\boldsymbol{V}_{\boldsymbol{\alpha}, \boldsymbol{\beta}} = \boldsymbol{\beta}^{\boldsymbol{\alpha}}$. Lemma 2.8 in (Wendland, 2004) implies that $\boldsymbol{V}$ is non-singular. Let $c_{n,D} = \left\| \boldsymbol{V}^{-1} \right\|_\infty$; note that $c_{n,D}$ depends only upon n, D.

Lemma 1. *Fix $D, n \in \mathbb{N}$, and let $\boldsymbol{\delta} \in \mathcal{B}$ be arbitrary. Then, there exist coefficients $\boldsymbol{a} \in \mathbb{R}^N$, $\|\boldsymbol{a}\|_1 \le c_{n,D}$, so that $\boldsymbol{x}^{\boldsymbol{\delta}} = \sum_{\boldsymbol{\beta} \in \mathcal{B}} a_{\boldsymbol{\beta}} (\langle \boldsymbol{\beta}, \boldsymbol{x} \rangle + 1)^n$, for all $\boldsymbol{x} \in \mathbb{R}^D$.*

Using the Lemma, we assume that the MLP $m : \mathbb{R}^{D+1} \to \mathbb{R}$ is two-layer, over-parameterized of the form $m(\boldsymbol{x}, 1)$ (i.e., a constant 1 plugged in an extra $D + 1$ coordinate). We consider training m with random initialization and gradient descent using data $(\boldsymbol{x}, \boldsymbol{x}^{\boldsymbol{\delta}}) \in \mathbb{R}^D \times \mathbb{R}$ where $\boldsymbol{x}$ is sampled i.i.d. from some distribution $\mathcal{D}$ over $\mathbb{R}^D$. Let $g : \mathbb{R}^{D+1} \to \mathbb{R}$ defined as $g(\boldsymbol{x}, x_{D+1}) = \sum_{\boldsymbol{\beta} \in \mathcal{B}} a_{\boldsymbol{\beta}} \left(\langle \boldsymbol{\beta}, \boldsymbol{x} \rangle + x_{D+1} \right)^n$, where $\boldsymbol{a} \in \mathbb{R}^N$ is according to Lemma 1. Then, the learning setup described above is equivalent to training the MLP $m(\boldsymbol{x}, x_{D+1})$ using data of the form $((\boldsymbol{x}, 1), g(\boldsymbol{x}, 1) = \boldsymbol{x}^{\boldsymbol{\delta}})$, where $(\boldsymbol{x}, 1)$ is sampled i.i.d. from a distribution $\mathcal{D}'$ over $\mathbb{R}^{D+1}$ concentrated on the hyperplane $x_{D+1} = 1$. Now by Corollary 6.2 from (Arora et al., 2019) $\boldsymbol{x}^{\boldsymbol{\delta}}$ is learnable by the MLP m and the sample complexity is bounded by

$$\mathcal{C}_{\mathcal{A}}(g, \epsilon, \delta) = \mathcal{O}\left(\frac{C_{n,D} + \log(1/\delta)}{\epsilon^2} \right),$$

where $C_{n,D} = \mathcal{O}((n^2 + 1)^{(n+1)/2} c_{n,D})$. The asymptotic behaviour of $c_{n,D}$ is out of scope for this paper, but in any case $C_{n,D}$ grows exponentially with the graph size. We can say however, that for a fixed graph size, and feature dimension, $C_{n,D}$ can be considered as a (very large) constant.

Discussion. The LRGA module is shown to be theoretically powerful when restricted to rich feature graphs and large rank κ. In practice, the edge structure is only partially manifested in the node features, and κ is maintained low for computational complexity. For these reasons LRGA complements GNNs that in turn transfers edge information to the node representation.

Table 4. Results of augmented GNNs with LRGA on ZINC .

Model	Model size $\sim 100K$	Model size $\sim 300K$
	MAE ± std	MAE ± std
LRGA + GCN	0.457 ± 0.004	0.433 ± 0.008
LRGA + GAT	0.438 ± 0.007	0.432 ± 0.016
LRGA + GIN	0.363 ± 0.010	0.355 ± 0.032

4. Experiments

We evaluated our method on various tasks from two benchmarks. We follow each benchmark evaluation protocol designed for a fair comparison of different models.

Baselines. We compare performance with the following baselines: *MLP*, *GCN* (Kipf and Welling, 2016), *GraphSAGE* (Hamilton et al., 2017), *GIN* (Xu et al., 2019a), *DiffPool* (Ying et al., 2018), *GAT* (Veličković et al., 2018), *MoNet* (Monti et al., 2017) and *GatedGCN* (Bresson and Laurent, 2017), *Node2Vec* (Grover and Leskovec, 2016) and *MATRIX FACTORIZATION* (Hu et al., 2020)

4.1. Benchmarking GNNs (Dwivedi et al., 2020)

Tables 2 and 3 summarize the results on this benchmark; LRGA combined with GatedGCN achieves SOTA performance in most of the datasets in the benchmark. To obey the parameter budget when LRGA is combined with GatedGCN we reduce the width of the GatedGCN layers. While improving SOTA for CLUSTER, PATTERN, CIFAR10, and MNIST, we found that LRGA did not improve GatedGCN on TSP and ZINC. To see if LRGA can improve GatedGCN with higher parameter budget, we enlarged the parameter budget to 2M and reevaluated all models on the ZINC dataset. As seen in table 3 our model improved SOTA by a large margin. We further explored the contribution of LRGA to other GNN architectures, see table 4. All models in the table were evaluated with the same augmented LRGA module size ($\kappa = 30$) and two versions for their own size: the original setting as appears in the benchmark (model size of ∼ 300K) versus a reduced model that fits the parameter budget (model size of ∼ 100K). Observing table 4, we see that LRGA improved all the GNNs considerably when augmented to GNNs without the 100K budget, while improving GCN, GAT, and GIN in the reduced 100K setting.

4.2. Link prediction OGB (Hu et al., 2020)

Table 1 summarizes the results on ogbl-ppa and ogbl-collab. It should be noted that the first two rows correspond to node embedding methods where the rest are GNNs. Augmenting GCN with LRGA achieves a major improvement on those datasets. Larger versions of LRGA+GCN achieve SOTA results on these datasets, while still using less parameters than node embedding methods.

References

Abbe, E. (2017). Community detection and stochastic block models: recent developments.

Arora, S., Du, S. S., Hu, W., Li, Z., and Wang, R. (2019). Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. *arXiv preprint arXiv:1901.08584*.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. (2018). Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*.

Battaglia, P. W., Pascanu, R., Lai, M., Rezende, D., and Kavukcuoglu, K. (2016). Interaction Networks for Learning about Objects, Relations and Physics.

Bresson, X. and Laurent, T. (2017). Residual Gated Graph Convnets. Technical report.

Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. (2014). Spectral networks and deep locally connected networks on graphs. In *2nd International Conference on Learning Representations, ICLR 2014*.

Defferrard, M., Bresson, X., and Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852.

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. (2015). Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232.

Dwivedi, V. P., Joshi, C. K., Laurent, T., Bengio, Y., and Bresson, X. (2020). Benchmarking graph neural networks.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural Message Passing for Quantum Chemistry. In *International Conference on Machine Learning*, pages 1263–1272.

Grover, A. and Leskovec, J. (2016). node2vec: Scalable feature learning for networks.

Hamilton, W. L., Ying, R., and Leskovec, J. (2017). Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, volume 2017-Decem, pages 1025–1035.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. (2020). Open Graph Benchmark: Datasets for Machine Learning on Graphs.

Joshi, C. K., Laurent, T., and Bresson, X. (2019). An efficient graph convolutional network technique for the travelling salesman problem.

Kipf, T. N. and Welling, M. (2016). Semi-Supervised Classification with Graph Convolutional Networks. *5th International Conference on Learning Representations, ICLR 2017*.

Knyazev, B., Taylor, G. W., and Amer, M. R. (2019). Understanding attention in graph neural networks. *CoRR*, abs/1905.02850.

Li, Y., Zemel, R., Brockschmidt, M., and Tarlow, D. (2016). Gated graph sequence neural networks. In *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*.

Maron, H., Ben-Hamu, H., Serviansky, H., and Lipman, Y. (2019a). Provably powerful graph networks. In *Advances in Neural Information Processing Systems 32*, pages 2156–2167. Curran Associates, Inc.

Maron, H., Ben-Hamu, H., Shamir, N., and Lipman, Y. (2019b). Invariant and equivariant graph networks. In *7th International Conference on Learning Representations, ICLR 2019*.

Monti, F., Boscaini, D., Masci, J., Rodola, E., Svoboda, J., and Bronstein, M. M. (2017). Geometric deep learning on graphs and manifolds using mixture model CNNs. In *Proc. CVPR*, volume 1, page 3.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. (2018). Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks. *arXiv preprint arXiv:1810.02244*.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. (2019). Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:4602–4609.

Niepert, M., Ahmed, M., and Kutzkov, K. (2016). Learning Convolutional Neural Networks for Graphs.

Vapnik, V. N. (1998). *Statistical Learning Theory*. Wiley-Interscience.

Veličković, P., Casanova, A., Liò, P., Cucurull, G., Romero, A., and Bengio, Y. (2018). Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018*.

Wendland, H. (2004). *Scattered data approximation*, volume 17. Cambridge university press.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2019a). How Powerful are Graph Neural Networks? In *International Conference on Learning Representations*.

Xu, K., Li, J., Zhang, M., Du, S. S., Kawarabayashi, K.-i., and Jegelka, S. (2019b). What Can Neural Networks reason About? Technical report.

Ying, R., You, J., Morris, C., Ren, X., Hamilton, W. L., and Leskovec, J. (2018). Hierarchical graph representation learning with differentiable pooling.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. (2017). Deep sets. In *Advances in Neural Information Processing Systems*, pages 3391–3401.

A. Proof of Proposition 1

Proof. Every graph function can be formulated as a function of the isomorphism type tensor $\mathbf{Y} \in \mathbb{R}^{n^2 \times d}$, and we will approximate such arbitrary continuous functions with GNN. Let $f : \mathbb{R}^{n^2 \times d} \to \mathbb{R}$ be a continuous invariant graph function (i.e., agnostic to ordering the graph nodes) defined over the isomorphism type tensors $\mathbb{R}^{n^2 \times d}$. Define $\hat{f}(\boldsymbol{X}) = f(\mathbf{Y})$, where $\mathbf{Y}$ is defined as in Definition 1. $\hat{f} : K \to \mathbb{R}$ is an invariant set function since it is a composition of invariant and equivariant functions (see e.g., (Maron et al., 2019b) for definition of equivariance); it is also continuous as a composition of continuous functions. Hence $\hat{f}$ can be approximated over K using DeepSets (Zaheer et al., 2017) (due to DeepSets universality). Since the GNN in (Battaglia et al., 2018) includes DeepSets as a particular case it can approximate $\hat{f}$ as-well. □

B. 2-FWL via polynomial kernels

In this section, we give a full characterization of feature maps, $\varphi_{\boldsymbol{\beta}}$, of the final polynomial kernel we use to formulate the 2-FWL algorithm. A key tool for the derivation of the final feature map is the multinomial theorem, which we state here in a slightly different form to fit our setting.

Multinomial theorem. Let us define a set of m variables $x_1y_1, \dots, x_my_m$ composed of products of corresponding x and y's. Then,

$$(x_1y_1 + \cdots + x_my_m)^n = \sum_{|\boldsymbol{\nu}|=n} \binom{n}{\boldsymbol{\nu}} \prod_{i=1}^{m} (x_iy_i)^{\nu_i}$$

where $\boldsymbol{\nu} \in \mathbb{N}_0^m$, and the notation $\binom{n}{\boldsymbol{\nu}} = \frac{n!}{\nu_1! \cdots \cdot \nu_m!}$. The sum is over all possible $\boldsymbol{\nu}$ which sum to n, in total $\binom{n+m-1}{m-1}$ elements.

Recall that we wish to compute $\mathbf{Y}^{\boldsymbol{\beta}}_{i,j}$ as shown in the paper:

$$\mathbf{Y}^{\boldsymbol{\beta}}_{i,j} = \prod_{k=1}^{d} \left\langle \boldsymbol{x}_i^{2k-1}, \boldsymbol{x}_j^{2k} \right\rangle^{\beta_k} = \prod_{k=1}^{d} \left\langle \varphi_{\beta_k}(\boldsymbol{x}_i^{2k-1}), \varphi_{\beta_k}(\boldsymbol{x}_j^{2k}) \right\rangle$$

$$= \prod_{k=1}^{d} \left\langle \varphi_{\beta_k}(\boldsymbol{x}_i^{\text{odd}}), \varphi_{\beta_k}(\boldsymbol{x}_j^{\text{even}}) \right\rangle = \left\langle \varphi_{\boldsymbol{\beta}}(\boldsymbol{x}_i^{\text{odd}}), \varphi_{\boldsymbol{\beta}}(\boldsymbol{x}_j^{\text{even}}) \right\rangle$$

We will now follow the equalities to derive the final feature map. The second equality is using the feature maps φ_{β_k} of the (homogeneous) polynomial kernels (Vapnik, 1998), $\langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle^{\beta_k}$, which can be derived from the multinomial theorem.

Suppose the dimensions of $\boldsymbol{X}^{2k-1}, \boldsymbol{X}^{2k}$ are $n \times D_k$ where $\sum_{k=1}^{d} 2D_k = D$. Then, φ_{β_k} consists of monomials of degree β_k of the form $\varphi_{\beta_k}(\boldsymbol{x})_{\boldsymbol{\nu}} = \sqrt{\binom{\beta_k}{\boldsymbol{\nu}}} \prod_{i=1}^{D_k} x_i^{\nu_i} = \sqrt{\binom{\beta_k}{\boldsymbol{\nu}}} \boldsymbol{x}^{\boldsymbol{\nu}}$, $|\boldsymbol{\nu}| = \beta_k$. In total the size of the feaure map φ_{β_k} is $\binom{\beta_k + D_k - 1}{D_k - 1}$.

The third equality is reformulating the feature maps φ_{β_k} on the vectors $\boldsymbol{x}_i^{\text{odd}} = \left[\boldsymbol{x}_i^1, \boldsymbol{x}_i^3, \dots, \boldsymbol{x}_i^{2d-1}\right] \in \mathbb{R}^{D/2}$, and $\boldsymbol{x}_i^{\text{even}} = \left[\boldsymbol{x}_i^2, \boldsymbol{x}_i^4, \dots, \boldsymbol{x}_i^{2d}\right] \in \mathbb{R}^{D/2}$.

The last equality is due to the closure of kernels to multiplication. The final feature map, which is the product kernel, is composed of all possible products of elements of the feature maps, i.e.,

$$\varphi_{\boldsymbol{\beta}}(\boldsymbol{x}) = \left(\prod_{k=1}^{d} \sqrt{\binom{\beta_k}{\boldsymbol{\nu}_k}} \boldsymbol{x}_k^{\boldsymbol{\nu}_k} \;\middle|\; |\boldsymbol{\nu}_j| = \beta_j, \; \forall j \in [d] \right),$$

where $\boldsymbol{x} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_d] \in \mathbb{R}^{D/2}$, and $\boldsymbol{x}_k \in \mathbb{R}^{D_k}$ for all $k \in [d]$. The size of the final feature map is $\prod_{k=1}^{d} \binom{\beta_k + D_k - 1}{D_k - 1} \leq N$ where $N = \binom{n+D}{D}$.

C. Bound on LRGA sample complexity

C.1. Proof of Lemma 1

Proof. Using the multinomial theorem we have: $(\langle \boldsymbol{\beta}, \boldsymbol{x} \rangle + 1)^n = \sum_{\boldsymbol{\alpha} \in \mathcal{B}} d_{\boldsymbol{\alpha}} \boldsymbol{\beta}^{\boldsymbol{\alpha}} \boldsymbol{x}^{\boldsymbol{\alpha}}$, where $d_{\boldsymbol{\alpha}}$ are positive multinomial coefficients. This equation defines a linear relation between the monomial basis $\boldsymbol{x}^{\boldsymbol{\delta}}$ and $(\langle \boldsymbol{\beta}, \boldsymbol{x} \rangle + 1)^n$, for $\boldsymbol{\beta} \in \mathcal{B}$. The matrix of this system is $\boldsymbol{V}$ multiplied by a positive diagonal matrix with $d_{\boldsymbol{\alpha}}$ on its diagonal. By inverting this matrix and solving this system for $\boldsymbol{x}^{\boldsymbol{\delta}}$ the lemma is proved. □

C.2. Derivation of sample complexity bound

Corollary 6.2 in (Arora et al., 2019) provides a bound on the sample complexity, denoted $\mathcal{C}_{\mathcal{A}}(g, \epsilon, \delta)$, of a polynomial $g : \mathbb{R}^D \to \mathbb{R}$ of the form

$$g(\boldsymbol{x}) = \sum_{j} a_j \left\langle \boldsymbol{\beta}_j, \boldsymbol{x} \right\rangle^{p_j}, \tag{8}$$

where $p_j \in \{1, 2, 4, 6, 8, \ldots\}$, $a_j \in \mathbb{R}$, $\boldsymbol{\beta}_j \in \mathbb{R}^D$; ϵ, δ are the relevant PAC learning constants, and $\mathcal{A}$ represents an over-parameterized, randomly initialized two-layer MLP trained with gradient descent:

$$\mathcal{C}_{\mathcal{A}}(g, \epsilon, \delta) = \mathcal{O}\left(\frac{\sum_j p_j \left|a_j\right| \left\|\boldsymbol{\beta}_j\right\|_2^{p_j} + \log(1/\delta)}{\epsilon^2}\right)$$

In our case $g : \mathbb{R}^{D+1} \to \mathbb{R}$ is defined as $g(\boldsymbol{x}, x_{D+1}) = \sum_{\boldsymbol{\beta} \in \mathcal{B}} a_{\boldsymbol{\beta}} \left(\langle \boldsymbol{\beta}, \boldsymbol{x} \rangle + x_{D+1}\right)^n$ where $\mathcal{B} = \left\{\boldsymbol{\beta} \in \mathbb{N}_0^D \mid |\boldsymbol{\beta}| \leq n\right\}$ and by Lemma 1 there exist $\boldsymbol{a}$ such that $g(\boldsymbol{x}, 1) = \boldsymbol{x}^{\boldsymbol{\delta}}$. The sample complexity bound expression by Corollary 6.2 is therefore:

$$\mathcal{C}_{\mathcal{A}}(g, \epsilon, \delta) = \mathcal{O}\left(\frac{\sum_{\boldsymbol{\beta} \in \mathcal{B}} n \left|a_{\boldsymbol{\beta}}\right| \left\|\hat{\boldsymbol{\beta}}\right\|_2^n + \log(1/\delta)}{\epsilon^2}\right)$$

when $\hat{\boldsymbol{\beta}} = (\boldsymbol{\beta}, 1)$.

Let us bound the first term in the numerator of the sample complexity expression:

$$\begin{aligned} &\sum_{\boldsymbol{\beta} \in \mathcal{B}} n \left|a_{\boldsymbol{\beta}}\right| \left\|\hat{\boldsymbol{\beta}}\right\|_2^n = n \cdot \sum_{\boldsymbol{\beta} \in \mathcal{B}} \left|a_{\boldsymbol{\beta}}\right| \left(\sum_{i=1}^{D} \beta_i^2 + 1\right)^{n/2} \\ &\leq n \cdot \left(n^2 + 1\right)^{n/2} \sum_{\boldsymbol{\beta} \in \mathcal{B}} \left|a_{\boldsymbol{\beta}}\right| \leq \left(n^2 + 1\right)^{(n+1)/2} c_{n,D} \end{aligned}$$

The first inequality is due to $\|\cdot\|_2 \leq \|\cdot\|_1$, the second is by Lemma 1 and uniting n into the main term. From the above, the bound follows.

Table 5. Summary of the benchmarking GNN and ogb link prediction Datasets

Dataset	#Graphs	#Nodes	Avg. Nodes	Avg. Edges	#Classes
ZINC	12K	9-37	23.16	49.83	-
CLUSTER	12K	40-190	117.20	4301.72	6
PATTERN	14K	50-180	117.47	4749.15	2
MNIST	70K	40-75	70.57	564.53	10
CIFAR10	60K	85-150	117.63	941.07	10
TSP	12K	50-500	275.76	6894.04	2
obgl-ppa	1	576,289	-	30,326,273	-
obgl-collab	1	235,868	-	1,285,465	-

D. Implementation Details

In this section we describe the datasets on which we performed our evaluation. In addition, we specify the hyperparameters for the experiments section in the paper. The rest of the model configurations are determined directly by the evaluation protocols defined by the benchmarks. It is worth noting that most of our experiments ran on a single Tesla V-100 GPU, if not stated otherwise. We performed our parameter search only on κ and d (except for CIFAR10 and MNIST were we searched over different dropout values), since the rest of the parameters were dictated by the evaluation protocol. The models sizes were restricted by the allowed parameter budget.

D.1. Benchmarking Graph Neural Networks (Dwivedi et al., 2020)

Datasets. This benchmark contains 6 main datasets :

(i) **ZINC**, a molecular graphs dataset with a graph regression task where each node represents an atom and each edge represents a bond. The regression target is a property known as the constrained solubility (with mean absolute error as evaluation metric). Additionally, the node features represent the atom's type (28 types) and the edge features represents the type of connection (4 types). The hyperparameters range which we used in our search was $\kappa \in \{20, 25, 30, 40\}$ and $d \in \{35, 40, 45, 50, 55\}$. For the reported results we used $\kappa = 30, d = 45$ and the averaged time for a single epoch (whole training) was 15.5 seconds (27.5 minutes).

(ii) **MNIST** and **CIFAR10**, the known image classification problem is converted to a graph classification task using Super-pixel representation (Knyazev et al., 2019), which represents small regions of homogeneous intensity as nodes. The edges in the graph are obtained by applying k-nearest neighbor algorithm on the nodes coordinates. Node features are a concatenation of the Super-pixel intensity (RGB for CIFAR10 and greyscale for MNIST) and its image coordinate. Edges features are the k-nearest distances. For the CIFAR10 and MNIST datasets our search range was $\kappa \in \{20, 25, 30\}$, $d = \{45, 50\}$ and $p \in \{0, 0.1, 0.2, 0.3, 0.5\}$. The chosen hyperparameters for the CIFAR10 dataset were $\kappa = 30, d = 45$ with additional dropout of $p = 0.1$. The averaged time for a single epoch (whole training) is 238.5 seconds (4.77 hours). We used the same hyperparameters for the MNIST dataset, besides the dropout which was changed to $p = 0.2$. Average time per epoch (whole training) is 197.69 seconds (3.84 hours).

(iii) **CLUSTER** and **PATTERN**, node classification tasks which aim to identify embedded node structures in stochastic block model graphs (Abbe, 2017). The goal of the task is to assign each node to the stochastic block it was originated from, while the structure of the graph is governed by two probabilities that define the inner-structure and cross-structure edges. A single representative from each block is assigned with an initial feature that indicates its block while the rest of the nodes have no features. We searched hyperparameters over the range $\kappa \in \{20, 25, 30, 40\}$ and $d \in \{35, 40, 45, 50, 55\}$. The hyperparameters for the CLUSTER dataset were $\kappa = 30, d = 45$. Average time per epoch (whole training) is 80.34 seconds (1.92 hours). For the PATTERN dataset we used $\kappa = 25, d = 50$. Averaged running time per epoch (whole training) is 153.83 seconds (3.476 hours), on a single Tesla P-100.

(iv) **TSP**, a link prediction task that tries to tackle the NP-hard classical Traveling Salesman Problem (Joshi et al., 2019). Given a 2D Euclidean graph the goal is to choose the edges that participate in the minimal edge weight tour of the graph. The evaluation metric for the task is F1 score for the positive class. Our hyperparameters search was in the range $\kappa \in \{20, 25, 30\}$ and $d \in \{45, 50, 55\}$, the results shown in the paper uses $\kappa = 20, d = 50$ and the averaged running time per epoch (whole training) is 166.02 seconds (20.5 hours), on a single Tesla P-100.

D.2. Link prediction datasets from the OGB benchmark (Hu et al., 2020)

Datasets. In order to provide a more complete evaluation of our model we also evaluate it on semi-supervised learning tasks of link prediction. We searched over the same hyperparameter range $\kappa \in \{25, 50, 100\}$, $d \in \{150, 256\}$ and used $\kappa = 50, d = 150$ in both tasks. The two datasets were:

(i) **ogbl-ppa**, an undirected unweighted graph. Nodes represent types of proteins and the edges signify biological connections between proteins. The initial node feature is a 58-dimensional one-hot-vector that indicates the origin specie of the protein. The learning task is to predict new connections between nodes. The train/validation/test split sizes are 21M/6M/3M . The evaluation metric is called Hits@K (Hu et al., 2020). Averaged running time was 4.5 minutes per epoch and 1.5 hours for the whole training.

(ii) **ogbl-collab**, is a graph that represents a network of collaborations between authors. Every author in the network is represented by a node and each collaboration is assigned with an edge. Initial node features are obtained by combining word embeddings of papers by that author (128-dimensional vector). Additionally, each collaboration is described by the year of collaboration and the number of collaborations in that year as a weight. The train/validation/test split sizes are 1.1M/60K/46K. Similarly to the previous dataset, the evaluation metric is Hits@K. Averaged running time was 5.22 seconds per epoch and 17.4 minutes for the whole training.