DEEPERGCN: ALL YOU NEED TO TRAIN DEEPER GCNS

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph Neural Networks (GNNs) have been drawing significant attention to the power of representation learning on graphs. Recent works developed frameworks to train very deep GNNs. Such works show impressive results in tasks like point cloud learning and protein interaction prediction. In this work, we study the performance of such deep models in large-scale graph datasets from the Open Graph Benchmark (OGB). In particular, we look at the effect of adequately choosing an aggregation function and its effect on final performance. Common choices of aggregation are *mean*, *max*, and *sum*. It has been shown that GNNs are sensitive to such aggregations when applied to different datasets. We systematically study this point on large-scale graphs and propose to alleviate it by introducing a novel *Generalized Aggregation Function*. Proposed aggregation functions extend beyond the commonly used ones. The generalized aggregation functions are fully differentiable, and thus their parameters can be learned in an end-to-end fashion. We show that deep residual GNNs equipped with generalized aggregation functions achieve state-of-the-art results in several benchmarks from OGB across tasks and domains.

1 INTRODUCTION

The rise of availability of non-Euclidean data (Bronstein et al., 2017) has recently shed interest into the topic of Graph Neural Networks (GNNs). GNNs provide powerful deep learning architectures for unstructured data, like point clouds and graphs. GNNs have already proven to be valuable in several applications including predicting individual relations in social networks (Tang & Liu, 2009), modelling proteins for drug discovery (Zitnik & Leskovec, 2017; Wale et al., 2008), enhancing predictions of recommendation engines (Monti et al., 2017b; Ying et al., 2018), and efficiently segmenting large point clouds (Wang et al., 2019; Li et al., 2019).

Graph convolutions in GNNs are based on the notion of message passing (Gilmer et al., 2017). To compute a new node feature at each GNN layer, information is aggregated from the node and its connected neighbors. Given the nature of graphs, aggregation functions must be permutation invariant. This property guarantees invariance/equivariance to isomorphic graphs (Battaglia et al., 2018; Xu et al., 2019b; Maron et al., 2019a). Popular choices for aggregation functions are *mean* (Kipf & Welling, 2016), *max* (Hamilton et al., 2017), and *sum* (Xu et al., 2019b). Recent works suggest different aggregations have different performance impact depending on the task. For example, *mean* and *sum* perform better in some node classification tasks such as classifying article subjects and detecting community (Xu et al., 2019b), while *max* is favorable for dealing with 3D point clouds (Qi et al., 2017; Wang et al., 2019). Despite *sum* is theoretically proven more powerful than *mean* and *max* with respect to graph isomorphism, it does not yield consistently better results on different datasets. The mechanisms to choose a suitable aggregation function are unclear. Most works rely on manual twists to choose aggregation functions.

In this work, we explore approaches to go beyond vanilla aggregation functions. In particular, we look at the effect of aggregation functions in performance. We unify aggregation functions by proposing a novel *Generalized Aggregation Function* (Figure 1) suited for graph convolutions. We show how our function covers all commonly used aggregations (*mean*, *max*, and *sum*), and its parameters can be tuned to learn customized functions for different tasks. Our novel aggregation is fully differentiable and can be learned in an end-to-end fashion in a deep GNN framework. In our experiments, we show the performance of baseline aggregations in various large-scale graph datasets. We then introduce our

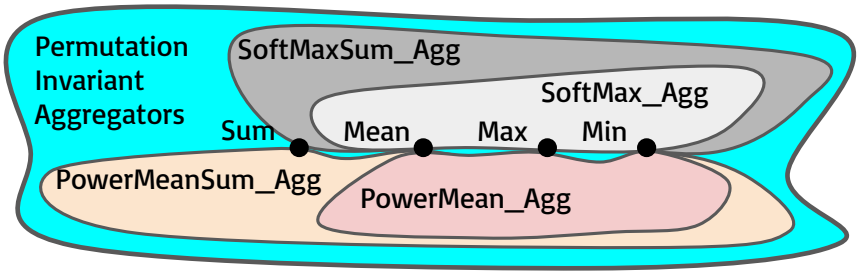


Figure 1: **Illustration of Generalized Message Aggregation Functions**

generalized aggregation and observe improved performance with the correct choice of aggregation parameters. Finally, we demonstrate how learning the parameters of our generalized aggregation, in an end-to-end fashion, leads to state-of-the-art performance in several OGB benchmarks. Our analysis indicates the choice of suitable aggregations is imperative to the performance of different tasks. A differentiable generalized aggregation function ensures the correct aggregation is used for each learning scenario.

We summarize our contributions as two-fold: (1) We propose a novel *Generalized Aggregation Function*. This new function is suitable for GNNs, as it enjoys a permutation invariant property. We show how our generalized aggregation covers commonly used functions such as *mean*, *max*, and *sum* in graph convolutions. We show its parameters can be tuned to improve performance on diverse graph learning tasks. Since this new function is fully differentiable, we show how its parameters can be learned in an end-to-end fashion. (2) We run extensive experiments on seven datasets from the Open Graph Benchmark (OGB). Our results show that combining pre-activated residual connections with our generalized aggregation network achieves state-of-the-art in several of these benchmarks.

2 Related Work

Graph Convolutional Networks (GNNs). GNN algorithms can be divided into two categories: spectral-based and spatial-based. Based on spectral graph theory, Bruna et al. (2013) firstly developed graph convolutions using the Fourier basis of a given graph in the spectral domain. Later, many methods proposed to apply improvements, extensions, and approximations on spectral-based GNNs (Kipf & Welling, 2016; Defferrard et al., 2016; Henaff et al., 2015; Levie et al., 2018; Li et al., 2018; Wu et al., 2019). Spatial-based GNNs (Scarselli et al., 2008; Hamilton et al., 2017; Monti et al., 2017a; Niepert et al., 2016; Gao et al., 2018; Xu et al., 2019b; Veličković et al., 2018) define graph convolution operations directly on the graph by aggregating information from neighbor nodes. Many sampling methods (Hamilton et al., 2017; Chen et al., 2018a;b; Li et al., 2018; Chiang et al., 2019; Zeng et al., 2020) have also been proposed to scale up GNNs on large-scale graphs.

Deep GNNs. Despite the rapid and fruitful progress of GNNs, most prior works employ shallow GNNs. Several works attempt different ways of training deeper GNNs (Rahimi et al., 2018; Xu et al., 2018). However, all these approaches are limited to 10 layers of depth, after which GNN performance would degrade because of vanishing gradient and over-smoothing (Li et al., 2018). Inspired by the merits of training deep CNN-based networks (He et al., 2016a; Huang et al., 2017; Yu & Koltun, 2016), DeepGCNs (Li et al., 2019; 2021) propose to train very deep GNNs (56 layers) by adapting residual connections, dense connections and dilated convolutions to GNNs. DeepGCN variants achieve state-of-the art results on S3DIS point cloud semantic segmentation (Armeni et al., 2017) and the PPI dataset. Many recent works focus on further addressing this phenomenon (Klicpera et al., 2019; Rong et al., 2020; Zhao & Akoglu, 2020; Chen et al., 2020; Gong et al., 2020; Rossi et al., 2020). In particular, Klicpera et al. (2019) propose a PageRank-based message passing mechanism involving the root node in the loop. Alternatively, DropEdge (Rong et al., 2020) randomly removes edges from the graph, and PairNorm (Zhao & Akoglu, 2020) develops a novel normalization layer. We find that the choice of aggregation functions also plays an important role in training deep GNNs. In this work, we thoroughly study the importance of aggregation functions in training deep GNNs.

Aggregation Functions for GNNs. GNNs update a node's feature vector by aggregating feature information from its neighbors in the graph. Many different neighborhood aggregation functions

that possess a permutation invariant property have been proposed (Hamilton et al., 2017; Veličković et al., 2018; Xu et al., 2019b). Specifically, Hamilton et al. (2017) examine mean, max, and LSTM aggregators, and they empirically find that max and LSTM achieve the best performance. Graph attention networks (GATs) (Veličković et al., 2018) employ the attention mechanism (Bahdanau et al., 2015) to obtain different and trainable weights for neighbor nodes by learning the attention between their feature vectors and that of the central node. Xu et al. (2019b) propose a Graph Isomorphism Network (GIN) with a sum aggregation and show its discriminative power is as powerful as Weisfeiler-Lehman (WL) test (Weisfeiler & Lehman, 1968). Corso et al. (2020) propose PNA by combining multiple aggregators with degree-scalers. In this work, we propose generalized message aggregation functions, a new family of aggregation functions, that generalizes vanilla aggregators including *mean*, *max* and *sum*. With the nature of differentiablity and continuity, generalized message aggregation functions provide a new perspective for designing GNN architectures.

3 Representation Learning on Graphs

Graph Representation. A graph $\mathcal{G}$ is usually defined as a tuple of two sets $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{ v_1, v_2, ..., v_N \}$ and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ are the sets of vertices and edges, respectively. If an edge $e_{ij} = (v_i, v_j) \in \mathcal{E}$ for an undirected graph, e_{ij} is an edge connecting vertices v_i and v_j; for a directed graph, e_{ij} is an edge directed from v_i to v_j. Usually, a vertex v and an edge e in the graph are associated with vertex features $\mathbf{h}_v \in \mathbb{R}^D$ and edge features $\mathbf{h}_e \in \mathbb{R}^C$ respectively.[1]

GNNs for Learning Graph Representation. We define a general graph representation learning operator $\mathcal{F}$, which takes as input a graph $\mathcal{G}$ and outputs a transformed graph $\mathcal{G}'$, *i.e.* $\mathcal{G}' = \mathcal{F}(\mathcal{G})$. The features or even the topology of the graph can be learned or updated after the transformation $\mathcal{F}$. Typical graph representation learning operators usually learn latent features or representations for graphs such as DeepWalk (Perozzi et al., 2014), Planetoid (Yang et al., 2016), Node2Vec (Grover & Leskovec, 2016), Chebyshev graph CNN (Defferrard et al., 2016), GCN (Kipf & Welling, 2016), Neural Message Passing Network (MPNN) (Gilmer et al., 2017), GraphSage (Hamilton et al., 2017), GAT (Veličković et al., 2018) and GIN (Xu et al., 2019b). In this work, we focus on the GNN family and its message passing framework (Gilmer et al., 2017; Battaglia et al., 2018). To be specific, message passing based on the GNN operator $\mathcal{F}$ operating on vertex $v \in \mathcal{V}$ at the l-th layer is defined as follows:

$$\mathbf{m}_{vu}^{(l)} = \boldsymbol{\rho}^{(l)}(\mathbf{h}_v^{(l)}, \mathbf{h}_u^{(l)}, \mathbf{h}_{e_{vu}}^{(l)}), \ \forall u \in \mathcal{N}(v) \tag{1}$$

$$\mathbf{m}_v^{(l)} = \boldsymbol{\zeta}^{(l)}(\{ \mathbf{m}_{vu}^{(l)} \mid u \in \mathcal{N}(v) \}) \tag{2}$$

$$\mathbf{h}_v^{(l+1)} = \boldsymbol{\phi}^{(l)}(\mathbf{h}_v^{(l)}, \mathbf{m}_v^{(l)}), \tag{3}$$

where $\boldsymbol{\rho}^{(l)}, \boldsymbol{\zeta}^{(l)}$, and $\boldsymbol{\phi}^{(l)}$ are all learnable or differentiable functions for *message construction*, *message aggregation*, and *vertex update* at the l-th layer, respectively. For simplicity, we only consider the case where vertex features are updated at each layer. It is straightforward to extend it to edge features. Message construction function $\boldsymbol{\rho}^{(l)}$ is applied to vertex features $\mathbf{h}_v^{(l)}$ of v, its neighbor's features $\mathbf{h}_u^{(l)}$, and the corresponding edge features $\mathbf{h}_{e_{vu}}$ to construct an individual message $\mathbf{m}_{vu}^{(l)}$ for each neighbor $u \in \mathcal{N}(v)$. Message aggregation function $\boldsymbol{\zeta}^{(l)}$ is commonly a permutation invariant set function that takes as input a countable unordered message set $\{ \mathbf{m}_{vu}^{(l)} \mid u \in \mathcal{N}(v) \}$, where $\mathbf{m}_{vu}^{(l)} \in \mathbb{R}^D$, and outputs a reduced or aggregated message $\mathbf{m}_v^{(l)} \in \mathbb{R}^D$. The permutation invariance of $\boldsymbol{\zeta}^{(l)}$ guarantees the invariance/equivariance to isomorphic graphs (Battaglia et al., 2018). $\boldsymbol{\zeta}^{(l)}$ can simply be a symmetric function such as *mean* (Kipf & Welling, 2016), *max* (Hamilton et al., 2017), or *sum* (Xu et al., 2019b). Vertex update function $\boldsymbol{\phi}^{(l)}$ combines the original vertex features $\mathbf{h}_v^{(l)}$ and the aggregated message $\mathbf{m}_v^{(l)}$ to obtain the transformed vertex features $\mathbf{h}_v^{(l+1)}$.

4 Beyond Mean, Max, and Sum Aggregation functions

Property 1 (Graph Isomorphic Equivariance)**.** *If a message aggregation function $\boldsymbol{\zeta}$ is permutation invariant to the message set $\{ \mathbf{m}_{vu} \mid u \in \mathcal{N}(v) \}$, then the message passing based GNN operator*

[1] In some cases, vertex features or edge features are absent.

$\mathcal{F}$ is equivariant to graph isomorphism, i.e. *for any isomorphic graphs $\mathcal{G}_1$ and $\mathcal{G}_2 = \sigma \star \mathcal{G}_1$, $\mathcal{F}(\mathcal{G}_2) = \sigma \star \mathcal{F}(\mathcal{G}_1)$, where $\star$ denotes applying a permutation operator σ on graphs.*

The invariance and equivariance properties on sets or GNNs have been discussed in many recent works. Zaheer et al. (2017) propose DeepSets based on permutation invariance and equivariance to deal with sets as inputs. Maron et al. (2019c) show the universality of invariant GNNs to any continuous invariant function. Keriven & Peyré (2019) further extend it to the equivariant case. Maron et al. (2019b) compose networks by proposing invariant or equivariant linear layers and show that their models are as powerful as any MPNN (Gilmer et al., 2017). In this work, we study permutation invariant functions of GNNs, which enjoy these proven properties.

4.1 Generalized Message Aggregation Functions

To embrace the properties of invariance and equivariance (Property 1), many works in the graph learning field tend to use simple permutation invariant functions like *mean* (Kipf & Welling, 2016), *max* (Hamilton et al., 2017) and *sum* (Xu et al., 2019b). Inspired by the Weisfeiler-Lehman (WL) graph isomorphism test (Weisfeiler & Lehman, 1968), Xu et al. (2019b) propose a theoretical framework and analyze the representational power of GNNs with *mean*, *max* and *sum* aggregators. Although *mean* and *max* aggregators are proven to be less powerful than *sum* according to the WL test in (Xu et al., 2019b), they are found to be quite effective in the tasks of node classification (Kipf & Welling, 2016; Hamilton et al., 2017) and 3D point cloud processing (Qi et al., 2017; Wang et al., 2019) To go beyond these simple aggregation functions and study their characteristics, we define generalized aggregation functions in the following.

Definition 2 (Generalized Message Aggregation Functions)**.** We define a generalized message aggregation function $\boldsymbol{\zeta}_{\boldsymbol{z}}(\cdot)$ as a function that is parameterized by a continuous variable $\boldsymbol{z}$ to produce a family of permutation invariant set functions, *i.e.* $\forall \boldsymbol{z}$, $\boldsymbol{\zeta}_{\boldsymbol{z}}(\cdot)$ is permutation invariant to the order of messages in the set $\{ \mathbf{m}_{vu} \mid u \in \mathcal{N}(v) \}$.

In order to subsume the popular *mean* and *max* aggregations into the generalized space, we further define *generalized mean-max aggregation* parameterized by a scalar for message aggregation.

Definition 3 (Generalized Mean-Max Aggregation)**.** If there exists a pair of x say x_1, x_2 such that for any message set $\lim_{x \to x_1} \boldsymbol{\zeta}_x(\cdot) = \text{Mean}(\cdot)$ [2] and $\lim_{x \to x_2} \boldsymbol{\zeta}_x(\cdot) = \text{Max}(\cdot)$, then $\boldsymbol{\zeta}_x(\cdot)$ is a generalized mean-max aggregation function.

The nice properties of generalized mean-max aggregation functions can be summarized as follows: **(1)** they provide a large family of permutation invariant aggregation functions; **(2)** they are continuous and differentiable in x and are potentially learnable; **(3)** it is possible to interpolate between x_1 and x_2 to find a better aggregator than *mean* and *max* for a given task. To empirically validate these properties, we propose two families of generalized mean-max aggregation functions based on Definition 3, namely *SoftMax aggregation* and *PowerMean aggregation*.

Proposition 4 (SoftMax Aggregation)**.** *Given any message set $\{ \mathbf{m}_{vu} \mid u \in \mathcal{N}(v) \}$, $\mathbf{m}_{vu} \in \mathbb{R}^D$, SoftMax_Agg$_\beta(\cdot)$ is a generalized mean-max aggregation function, where SoftMax_Agg$_\beta(\cdot) = \sum_{u \in \mathcal{N}(v)} \frac{\exp(\beta \mathbf{m}_{vu})}{\sum_{i \in \mathcal{N}(v)} \exp(\beta \mathbf{m}_{vi})} \cdot \mathbf{m}_{vu}$. Here β is a continuous variable called an inverse temperature.*

The SoftMax function with a temperature has been studied in many machine learning areas, *e.g.* Energy-Based Learning (LeCun et al., 2006), Knowledge Distillation (Hinton et al., 2015) and Reinforcement Learning (Gao & Pavel, 2017). Here, for low inverse temperatures β, SoftMax_Agg$_\beta(\cdot)$ behaves like a mean aggregation. For high inverse temperatures, it approaches a max aggregation. Formally, $\lim_{\beta \to 0}$SoftMax_Agg$_\beta(\cdot) = \text{Mean}(\cdot)$ and $\lim_{\beta \to \infty}$SoftMax_Agg$_\beta(\cdot) = \text{Max}(\cdot)$. It can be regarded as a weighted summation that depends on the inverse temperature β and the values of the elements themselves. The full proof of Proposition 4 is in the Appendix.

Proposition 5 (PowerMean Aggregation)**.** *Given any message set $\{ \mathbf{m}_{vu} \mid u \in \mathcal{N}(v) \}$, $\mathbf{m}_{vu} \in \mathbb{R}^D_+$, PowerMean_Agg$_p(\cdot)$ is a generalized mean-max aggregation function, where PowerMean_Agg$_p(\cdot) = (\frac{1}{|\mathcal{N}(v)|} \sum_{u \in \mathcal{N}(v)} \mathbf{m}_{vu}^p)^{1/p}$. Here, p is a non-zero, continuous variable denoting the p-th power.*

[2] Mean$(\cdot)$ denotes the arithmetic mean.

Quasi-arithmetic mean (Kolmogorov & Castelnuovo, 1930) was proposed to unify the family of mean functions. Power mean is one member of the Quasi-arithmetic mean family. It is a generalized mean function that includes harmonic mean, geometric mean, arithmetic mean, and quadratic mean. The main difference between Proposition 4 and 5 is that Proposition 5 only holds when message features are all positive, *i.e.* $\mathbf{m}_{vu} \in \mathbb{R}_+^D$. In particular, we have PowerMean_Agg$_{p=1}(\cdot) = \text{Mean}(\cdot)$ and $\lim_{p\to\infty}$PowerMean_Agg$_p(\cdot) = \text{Max}(\cdot)$. PowerMean_Agg$_p(\cdot)$ becomes the harmonic or the geometric mean aggregation when $p = -1$ or $p \to 0$, respectively. See the Appendix for the proof.

To enhance expressive power according to the WL test (Xu et al., 2019b), we generalize the function space to cover the *sum* aggregator by introducing another control variable on the degree of vertices.

Proposition 6 (Generalized Mean-Max-Sum Aggregation)**.** *Given any generalized mean-max aggregation function $\boldsymbol{\zeta}_x(\cdot)$, we can generalize the function to cover* sum *by combining it with the degree of vertices. For instance, by introducing a variable y, we can compose a generalized mean-max-sum aggregation function as $\left|\mathcal{N}(v)\right|^y \cdot \boldsymbol{\zeta}_x(\cdot)$. We can observe that the function becomes a Sum aggregation when $\boldsymbol{\zeta}_x(\cdot)$ is a Mean aggregation and $y = 1$. By composing with SoftMax aggregation and PowerMean aggregation, we obtain SoftMaxSum_Agg$_{(\beta,y)}(\cdot)$ and PowerMeanSum_Agg$_{(p,y)}(\cdot)$ aggregation functions, respectively.*

4.2 GENERALIZED AGGREGATION NETWORKS (GEN)

Generalized Message Passing Layer. Based on the Propositions above, we construct a simple message passing based GNN network that satisfies the conditions in Proposition 4 and 5. The key idea is to keep all the message features to be positive, so that generalized mean-max aggregation functions (SoftMax_Agg$_\beta(\cdot)$ and PowerMean_Agg$_p(\cdot)$) can be applied. We define the message construction function $\boldsymbol{\rho}^{(l)}$ as follows:

$$\mathbf{m}_{vu}^{(l)} = \boldsymbol{\rho}^{(l)}(\mathbf{h}_v^{(l)}, \mathbf{h}_u^{(l)}, \mathbf{h}_{e_{vu}}^{(l)}) = \text{ReLU}(\mathbf{h}_u^{(l)} + \mathbb{1}(\mathbf{h}_{e_{vu}}^{(l)} \neq \text{None}) \cdot \mathbf{h}_{e_{vu}}^{(l)}) + \epsilon,\ \forall u \in \mathcal{N}(v) \qquad (4)$$

where ReLU$(\cdot)$ is a rectified linear unit (Nair & Hinton, 2010) that outputs values to be greater or equal to zero, $\mathbb{1}(\cdot)$ is an indicator function being 1 when edge features exist otherwise 0, and ϵ is a small positive constant chosen to be 10^{-7}. As the conditions are satisfied, we can choose the message aggregation function $\boldsymbol{\zeta}^{(l)}(\cdot)$ to be either SoftMax_Agg$_\beta(\cdot)$, PowerMean_Agg$_p(\cdot)$, SoftMaxSum_Agg$_{(\beta,y)}(\cdot)$, or PowerMeanSum_Agg$_{(p,y)}(\cdot)$. As for the vertex update function $\boldsymbol{\phi}^{(l)}$, we use a simple multi-layer perceptron, where $\boldsymbol{\phi}^{(l)} = \text{MLP}(\mathbf{h}_v^{(l)} + \mathbf{m}_v^{(l)})$.

Skip Connections and Normalization. Skip connections and normalization techniques are important to train deep GNNs. Li et al. (2019) propose residual GNN blocks with components following the ordering: GraphConv $\to$ Normalization $\to$ ReLU $\to$ Addition. He et al. (2016b) studied the effect of ordering of ResNet components in CNNs, showing its importance. As recommended in their paper, the output range of the residual function should be $(-\infty, +\infty)$. Activation functions such as ReLU before addition may impede the representational power of deep models. Therefore, we adopt a pre-activation variant of residual connections for GNNs, which follows the ordering: Normalization $\to$ ReLU $\to$ GraphConv $\to$ Addition. Empirically, we find that the pre-activation version performs better. In our architectures, normalization methods such as BatchNorm (Ioffe & Szegedy, 2015) or LayerNorm (Ba et al., 2016) are applied to normalize vertex features.

5 EXPERIMENTS

We propose *GENeralized Aggregation Networks* (GEN) equipped with generalized message aggregators. To evaluate the effectiveness of these aggregators, we perform extensive experiments on the *Open Graph Benchmark* (OGB) (Hu et al., 2020), which includes a diverse set of challenging and large-scale tasks and datasets. We first conduct a comprehensive ablation study on the task of node property prediction on *ogbn-proteins* and *ogbn-arxiv* datasets. Then, we apply our GEN framework on the node property prediction dataset (*ogbn-products*), three graph property prediction datasets (*ogbg-molhiv*, *ogbg-molpcba* and *ogbg-ppa*), and one link property prediction dataset (*ogbl-collab*).

5.1 EXPERIMENTAL SETUP

ResGCN+. The PlainGCN model stacks GCNs from 3 layers to 112 layers without skip connections. Each GCN layer uses the same message passing operator as in GEN except the aggregation function is

replaced by Sum$(\cdot)$, Mean$(\cdot)$, or Max$(\cdot)$ aggregation. LayerNorm or BatchNorm is used in every layer before the ReLU activation function. Similar to Li et al. (2019), we construct the ResGCN model by adding residual connections to PlainGCN following the ordering: GraphGonv $\rightarrow$ Normalization $\rightarrow$ ReLU $\rightarrow$ Addition. We further present a pre-activation version of ResGCN by changing the order of components in residual blocks to Normalization $\rightarrow$ ReLU $\rightarrow$ GraphGonv $\rightarrow$ Addition. We denote this as ResGCN+ to differentiate it from ResGCN.

ResGEN. The ResGEN models are designed using the message passing functions described in Section 4.2. The only difference between ResGEN and ResGCN+ is that generalized message aggregators are used instead of Sum$(\cdot)$, Mean$(\cdot)$, or Max$(\cdot)$. For simplicity, we study generalized mean-max aggregators (*i.e*. SoftMax_Agg$_\beta(\cdot)$ and PowerMean_Agg$_p(\cdot)$) which are parameterized by only one scalar. To explore the characteristics of the generalized message aggregators, we initialize them with different hyper-parameters. Here, we initialize the values of β to 10^n, where $n \in \{-3, -2, -1, 0, 1, 2, 3, 4\}$ and p to $\{-1, 10^{-3}, 1, 2, 3, 4, 5, 10\}$.

DyResGEN. In contrast to ResGEN, DyResGEN learns variables β, p or y *dynamically* for every layer at every gradient descent step. By learning these variables, we avoid the need to painstakingly search for the best hyper-parameters. In doing so, DyResGEN can learn aggregation functions that adapt to the training process and the dataset. We study the potential of learning these variables for our proposed aggregators: SoftMax_Agg$_\beta(\cdot)$, PowerMean_Agg$_p(\cdot)$, SoftMaxSum_Agg$_{(\beta,y)}(\cdot)$, and PowerMeanSum_Agg$_{(p,y)}(\cdot)$.

Datasets. Traditional graph datasets have been shown limited and unable to provide reliable evaluation and rigorous comparison among methods (Hu et al., 2020; Dwivedi et al., 2020). Reasons include their small-scale nature, non-negligible duplication or leakage rates, unrealistic data splits, *etc*. Consequently, we conduct our experiments on the recently released datasets of Open Graph Benchmark (OGB) (Hu et al., 2020), which overcome the main drawbacks of commonly used datasets and thus are much more realistic and challenging. OGB datasets cover a variety of real-world applications and span several important domains ranging from social and information networks to biological networks, molecular graphs, and knowledge graphs. They also span a variety of prediction tasks at the level of nodes, graphs, and links/edges. In this work, experiments are performed on three OGB datasets for node property prediction, three OGB datasets for graph property prediction, and one OGB dataset for link property prediction. We introduce these seven datasets briefly in Appendix. More detailed information about OGB datasets can be found in (Hu et al., 2020).

Implementation Details. We first perform ablation studies on the ogbn-proteins and ogbn-arxiv datasets. Then, we evaluate our model on the other datasets and compare the performances with state-of-the-art (SOTA) methods. Since the ogbn-proteins dataset is very dense and comparably large, full-batch training is infeasible when considering very deep GCNs. We simply apply a random partition to generate batches for both mini-batch training and test. We set the number of partitions to 10 for training and 5 for test, and we set the batch size to 1 subgraph. In comparison, the ogbn-arxiv dataset is relatively small, so we conduct experiments via full batch training and test in this case.

5.2 Analyses and Ablation Studies

Depth & Residual connections. We conduct an ablation study on ogbn-protein to show the effect of pre-activation residual connections. Experiments in Figure 2 show that residual connections significantly improve the training dynamic of deep GCN models on ogbn-proteins. PlainGCN without skip connections suffers from vanishing gradient and does not gain any improvement from increasing depth. More prominent gains can be observed in ResGCN+ compared to ResGCN as models go deeper. Notably, ResGCN+ reaches smallest training loss with 112 layers.This validates the effectiveness of pre-activation residual connections. Note that all models in this ablation study use the Max aggregation function.

Effect of Aggregators in Training Deep GNNs. Although pre-activation residual connections alleviate the effect of vanishing gradients and enable the training of deep GCNs (see Appendix 5.2), the choice of aggregation function is crucial to performance. In Figure 3, we study how vanilla aggregators (*i.e*. Sum$(\cdot)$, Mean$(\cdot)$ and Max$(\cdot)$) behave on ogbn-proteins and ogbn-arxiv. We find that the aggregators perform inconsistently among different datasets and cause significant gaps in performance. For instance, Max$(\cdot)$ outperforms the other two by a large margin ($\sim 1\%$) for all network depths on ogbn-proteins, but reaches unsatisfactory results ($< 70\%$) and even becomes worse with depth increasing on ogbn-arxiv. Mean$(\cdot)$ performs the worst on ogbn-proteins, but

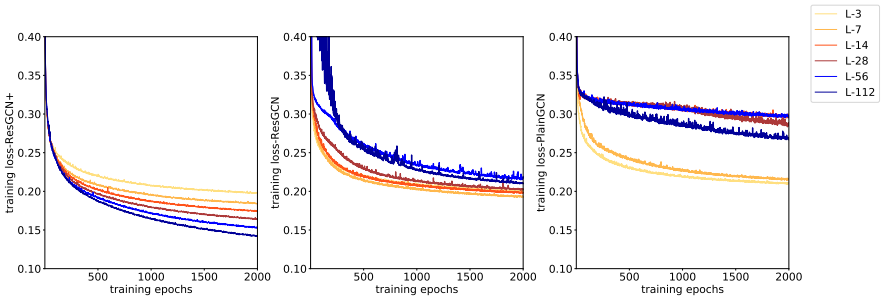
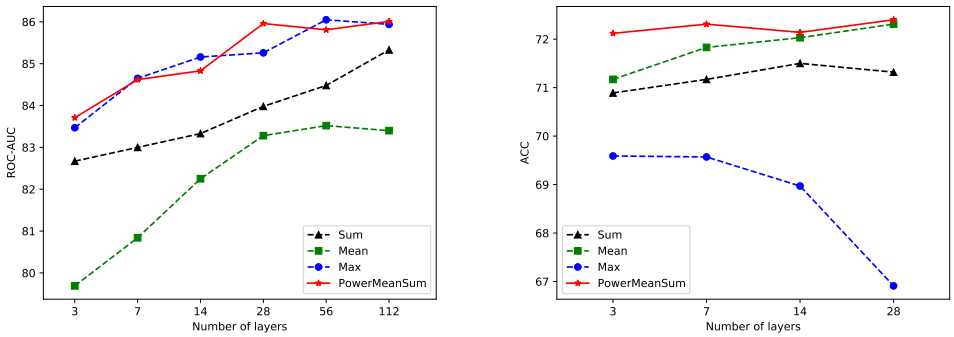


Figure 2: **Training losses of ResGCN+ and ResGCN, PlainGCN**



(a) different aggregators on the obgn-protein dataset. (b) different aggregations on the obgn-arxiv dataset.

Figure 3: **Vanilla aggregators perform differently and inconsistently on different datasets.** *e.g.* while Mean$(\cdot)$ performing the worst on obgn-protein, it achieves the best accuracy among all the vanilla aggregators on obgn-arxiv. The Max$(\cdot)$ achieves superb results on obgn-proteins but the worst resutls on obgn-arxiv. Our proposed generalized aggregator (PowerMeanSum) achieves consistently promising results across different datasets.

the best (72.31%) with 28 layers on ogbn-arxiv. To address this issue, we propose differentiable generalized aggregtion functions which can be learned to be adaptive to different datasets. We also explore the characteristics ResGEN with generalized message aggregators SoftMax_Agg$(\cdot)$ and PowerMean_Agg$(\cdot)$ by tuning the parameters β and p. Results show that better generalized mean-max aggregators exist beyond mean and max. Please refer to the Appendix for more details.

Table 1: **Ablation studies of aggregators on the ogbn-proteins dataset.** Proposed generalized message aggregators with learnable parameters outperforms the fixed parameters version as well as the vanilla aggregators overall.

Aggregation	Sum	Mean	Max	SoftMax		SoftMaxSum		PowerMean		PowerMeanSum	
#Layers	-	-	-	Fixed	Learned	Fixed	Learned	Fixed	Learned	Fixed	Learned
3	82.67	79.69	83.47	81.69	83.42	83.06	83.55	78.52	82.25	81.70	**83.71**
7	83.00	80.84	84.65	83.85	**84.81**	84.71	84.73	81.02	84.14	83.23	84.62
14	83.33	82.25	85.16	84.39	**85.29**	84.77	85.06	82.45	85.04	83.96	84.83
28	83.98	83.28	85.26	85.08	85.51	85.64	85.69	82.58	85.04	84.59	**85.96**
56	84.48	83.52	86.05	85.76	**86.12**	85.63	85.73	83.49	85.27	85.37	85.81
112	85.33	83.40	85.94	85.77	**86.15**	86.11	86.13	83.92	85.60	85.71	86.01
avg.	83.80	82.16	85.09	84.42	**85.22**	84.99	85.15	82.00	84.56	84.09	85.16

Learning Dynamic Aggregators. Trying out every possible aggregator or searching hyper-parameters is computationally expensive and time consuming. Therefore, we propose DyResGEN to explore the potential of learning dynamic aggregators by learning the parameters β, p, and even y within GEN. Table 1 reports the results of learning β, $\beta\&y$, p and $p\&y$ for SoftMax_Agg$(\cdot)$, SoftMaxSum_Agg$(\cdot)$, PowerMean_Agg$(\cdot)$ and PowerMeanSum_Agg$(\cdot)$ respectively. In practice, y is bounded from 0 to 1 by a Sigmoid function. In all experiments, we initialize the values of β, p

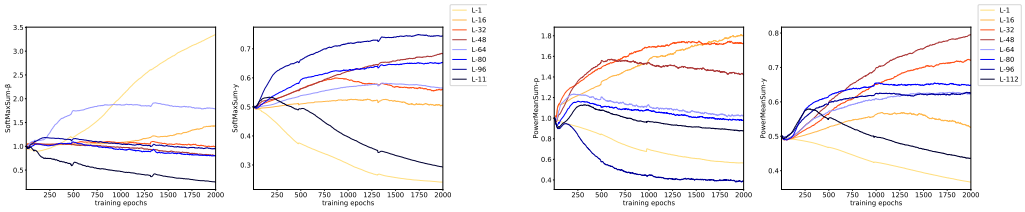
to 1 and y to 0.5 at the beginning of training. In order to show the improvement of the learning process, we also ablate experiments with fixed initial values. We denote aggregators with fixed initial values as *Fixed* and learned aggregators as *Learned*. We see that learning these variables consistently boosts the average performances of all the learned aggregators compared to the fixed initialized counterparts, which shows the effectiveness of learning adaptive aggregators. In particular, when β is learned, DyResGEN-SoftMax achieves 86.15% at 112 layers. We observe that DyResGEN-SoftMax outperforms the best ResGEN-SoftMax ($\beta = 10^4$) in terms of the average performance (85.22% *v.s.* 85.17%). Interesting, we find generalizing the *sum* aggregation with PowerMean significantly improve the average performance from 84.56% to 85.16%. In practice, we find SoftMax_Agg and PowerMeanSum_Agg performs better.

Learning SoftMax_Agg on More Datasets. In Figure 3 and Table 1 we show that proposed generalized message aggregators outperforms the vanilla ones in node property prediction datasets (ogbn-protein and ogbn-arxiv). Here, Table 2 presents more ablations to show the benefit of generalized message aggregators on the graph property prediction dataset (ogbg-molhiv) and the link property prediction datast (ogbl-collab). SoftMax_Agg$_\beta(\cdot)$ with learnable β achieves better performance compared to the same models using vanilla aggregators.

Analysis of DyResGEN. We provide more analysis on the learning dynamic of DyResGEN. The experimental results of DyResGEN in this section are obtained on ogbn-proteins dataset. We visualize the learning dynamic of learnable parameters β, p and s of 112-layer DyResGEN with SoftMaxSum_Agg$_{(\beta,y)}(\cdot)$ aggregator and PowerMeanSum_Agg$_{(p,y)}(\cdot)$ aggregator respectively. Learnable parameters β and p are initialized as 1 and y are initialized as 0.5. Dropout with a rate of 0.1 is used for each layer to prevent over-fitting. The learning curves of parameters of SoftMaxSum_Agg are shown in Figure 4a. We observe that both β and y change dynamically during the training. The β and y parameters of some layers tend to be stable after 1000 training epochs. Exceptionally, the 1-st layer learns a β increasingly from 1 to 3.3 which learns a smaller $y \approx 0.1$ which make SoftMaxSum_Agg$_{(\beta,y)}(\cdot)$ behave more like a Max aggregation at the 1-th layer. PowerMeanSum_Agg$_p(\cdot)$ aggregator also demonstrates a similar phenomena on learning y in Figure 4b. The learned y of the 1-st layer and the last layer trends to be smaller than the initial value.

Table 2: **Ablations of SoftMax_Agg against the vanilla ones on ogbg-molhiv and ogbl-collab.**

ROC-AUC	Sum	Mean	Max	SoftMax
ogbg-molhiv	76.84 ± 0.86	77.53 ± 1.59	78.71 ± 1.40	**78.87 ± 1.24**
Hits@50	Sum	Mean	Max	SoftMax
ogbl-collab	42.55 ± 1.29	51.08 ± 0.62	51.75 ± 0.76	**52.73 ± 0.47**



(a) SoftMaxSum_Agg$_{(\beta,y)}(\cdot)$ (b) PowerMeanSum_Agg$_{(p,y)}(\cdot)$

Figure 4: **Learning curves of 112-layer DyResGEN with SoftMaxSum and PowerMeanSum**

5.3 Comparison with SOTA

In this section, we apply the proposed GENeralized Aggregation Networks (GEN) with SoftMax aggregators to seven OGB datasets (ogbn-proteins, ogbn-arxiv, ogbn-products, ogbg-molhiv, ogbg-molpcba, ogbg-ppa and ogbl-collab) across various tasks of node classification, link prediction, and graph classification in Table 3. We apply our GEN models to the mentioned seven OGB datasets and compare results with the published GNN methods with official implementation posted on OGB Learderboard (See Table 3). The methods include Deepwalk (Perozzi et al., 2014), GCN (Kipf & Welling, 2016), GraphSAGE (Hamilton et al., 2017), GIN (Xu et al., 2019b), GIN or GCN with virtual nodes, JKNet (Xu et al., 2019a), GaAN (Zhang et al., 2018), GatedGCN (Bresson & Laurent, 2018), GAT (Veličković et al., 2018), HIMP (Fey et al., 2020), GCNII (Ming Chen et al., 2020), DAGNN (Liu et al., 2020a), GraphZoom (Deng et al., 2020), GeniePath-BS (Liu et al., 2020b) and

PNA (Corso et al., 2020). The provided results on each dataset are obtained by averaging the results from 10 independent runs. It is clear that our proposed GNN models outperform SOTA on all four datasets. In particular, our models significantly outperform previous SOTA methods on ogbn-proteins and ogbg-ppa by 7.91% and 6.75%, respectively. In terms of the performance on ogbn-arxiv, our model is better than JKNet, DAGNN but slight worse than GCNII. However, GCNII uses 4 times more parameters compared to our model (491,176 (Ours) *vs*. 2,148,648 (GCNII)). We try to adapt the official implementation of GCNII to ogbn-products but get out-of-memory error due to the high GPU memory consumption of GCNII. A comparison of ResGEN (SoftMax) *vs*. GAT w/ skip connections is shown on ogbn-products. ResGEN outputforms GAT w/ skip connections by 2.19% with only 33% number of parameters (253,743 (Ours) *vs*. 751,574 (GAT)). Notably, our method is the only one that shows promising results across various tasks and domains. More experimental details including datasets, hyper-parameters and implementation can be found in the Appendix.

Table 3: **Comparisons with SOTA on seven OGB datasets.** * denotes that virtual nodes are used.

	GraphSAGE	GCN	GaAN	GeniePath-BS			Ours
ogbn-proteins	77.68 ± 0.20	72.51 ± 0.35	78.03 ± 0.73	78.25 ± 0.35			**86.16 ± 0.16**
	GraphSAGE	GCN	GaAN	GCNII	JKNet	DAGNN	
ogbn-arxiv	71.49 ± 0.27	71.74 ± 0.29	71.97 ± 0.24	**72.74 ± 0.16**	72.19 ± 0.21	72.09 ± 0.25	72.32 ± 0.27
	GraphSAGE	GCN	ClusterGCN	GraphSAINT	GAT	GraphZoom	
ogbn-products	78.29 ± 0.16	75.64 ± 0.21	78.97 ± 0.33	80.27 ± 0.26	79.45 ± 0.59	74.06 ± 0.26	**81.64 ± 0.30**
	GIN	GCN	GIN*	GCN*	HIMP	PNA	
ogbg-molhiv	75.58 ± 1.40	76.06 ± 0.97	77.07 ± 1.49	75.99 ± 1.19	78.80 ± 0.82	**79.05 ± 1.32**	78.87 ± 1.24
ogbg-molpcba	22.66 ± 0.28	20.20 ± 0.24	27.03 ± 0.23	24.24 ± 0.34		**28.38 ± 0.35**	27.81 ± 0.38*
ogbg-ppa	68.92 ± 1.00	68.39 ± 0.84	70.37 ± 1.07	68.57 ± 0.61			**77.12 ± 0.71**
	GraphSAGE	GCN	DeepWalk				
ogbl-collab	48.10 ± 0.81	44.75 ± 1.07	50.37 ± 0.34				**52.73 ± 0.47**

Ablation on small models. In the Table 3, we show deep GNNs equipping with our aggregators can outperform the state-of-the-art methods. However, deep GNNs use more parameters than the shallow ones. Here, for a more fair comparison, we conduct an ablation study on the ogbn-proteins dataset by reducing the number of parameters of our model. Our original model has 64 hidden channels and 112 layers and uses SoftMax aggregators. We further create two compact models with fewer parameters than GCN and GraphSAGE by reducing the hidden sizes and the number of layers. Ours (Small-1) has 32 hidden channels and 14 layers. Ours (Small-2) has 16 hidden channels and 56 layers. Table 4 shows that our compact models still outperform GCN and GraphSAGE by a large margin (>5%).

6 Conclusion and Future Work

Table 4: **Ablation on Model Sizes on ogbn-proteins.**

Model	Params	Test ROC-AUC
GCN	96,880	72.51 ± 0.35
GraphSAGE	193,136	77.68 ± 0.20
Ours (Original)	2,374,568	86.16 ± 0.16
Ours (Small-1)	80,446	83.50 ± 0.30
Ours (Small-2)	82,888	82.69 ± 0.21

In this work, we identified the choice of aggregation functions is crucial to the performance of deep GNNs. We then proposed a differentiable generalized message aggregation function, which defines a family of permutation invariant aggregators. Systematic analysis and experiments show that our proposed function can yield better generalized aggregators that not only cover the widely used *mean*, *max* and *sum* but also go beyond them. We insert our generalized message aggregation functions into the proposed pre-activation residual GCN network and present GENeralized Aggregation Network (GEN). Empirically, we show the effectiveness of training our proposed deep GEN models, whereby we set a new SOTA on several datasets of the challenging Open Graph Benchmark. We believe the definition of such a generalized aggregation function provides a new view to the design of aggregation functions for GNNs. Here we discuss some interesting directions as follows: (1) Can we learn the parameters of generalized aggregation functions with a mete-learning method such as MAML (Finn et al., 2017)? (2) What is the expressive power generalized mean-max-sum aggregation functions with respect to Weisfeiler-Lehman test (Xu et al., 2019b)? (3) Can we design Principal Neighbourhood Aggregation (PNA) (Corso et al., 2020) by combining multiple learnable aggregators of generalized aggregation functions?

REFERENCES

I. Armeni, A. Sax, A. R. Zamir, and S. Savarese. Joint 2D-3D-Semantic Data for Indoor Scene Understanding. *ArXiv e-prints*, February 2017.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations*, 2015.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Xavier Bresson and Thomas Laurent. An experimental study of neural networks for variable graphs. In *International Conference on Learning Representations Workshop*, 2018.

Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction. In *International Conference on Machine Learning*, pp. 941–949, 2018a.

Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018b.

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. *arXiv preprint arXiv:2007.02133*, 2020.

Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 257–266, 2019.

Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *arXiv preprint arXiv:2004.05718*, 2020.

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

Chenhui Deng, Zhiqiang Zhao, Yongyu Wang, Zhiru Zhang, and Zhuo Feng. Graphzoom: A multi-level spectral approach for accurate and scalable graph embedding. In *International Conference on Learning Representations*, 2020.

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

M. Fey, J. G. Yuen, and F. Weichert. Hierarchical inter-message passing for learning on molecular graphs. In *ICML Graph Representation Learning and Beyond (GRL+) Workhop*, 2020.

Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017.

Bolin Gao and Lacra Pavel. On the properties of the softmax function with application in game theory and reinforcement learning. *arXiv preprint arXiv:1704.00805*, 2017.

Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. Large-scale learnable graph convolutional networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1416–1424, 2018.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

Shunwang Gong, Mehdi Bahri, Michael M Bronstein, and Stefanos Zafeiriou. Geometrically principled connections in graph neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11415–11424, 2020.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864, 2016.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pp. 1024–1034, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pp. 630–645. Springer, 2016b.

Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Nicolas Keriven and Gabriel Peyré. Universal invariant and equivariant graph neural networks. In *Advances in Neural Information Processing Systems*, pp. 7090–7099, 2019.

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

Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *International Conference on Learning Representations (ICLR)*, 2019.

Andreui Nikolaevich Kolmogorov and Guido Castelnuovo. *Sur la notion de la moyenne*. G. Bardi, tip. della R. Accad. dei Lincei, 1930.

Yann LeCun, Sumit Chopra, Raia Hadsell, M Ranzato, and F Huang. A tutorial on energy-based learning. *Predicting structured data*, 1(0), 2006.

Ron Levie, Federico Monti, Xavier Bresson, and Michael M Bronstein. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Transactions on Signal Processing*, 67(1):97–109, 2018.

Guohao Li, Matthias Müller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *The IEEE International Conference on Computer Vision (ICCV)*, 2019.

Guohao Li, Matthias Müller, Guocheng Qian, Itzel Carolina Delgadillo Perez, Abdulellah Abualshour, Ali Kassem Thabet, and Bernard Ghanem. Deepgcns: Making gcns go as deep as cnns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

Q. Li, Z. Han, and X.-M. Wu. Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning. In *The Thirty-Second AAAI Conference on Artificial Intelligence*. AAAI, 2018.

Ruoyu Li, Sheng Wang, Feiyun Zhu, and Junzhou Huang. Adaptive graph convolutional neural networks. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2020a.

Ziqi Liu, Zhengwei Wu, Zhiqiang Zhang, Jun Zhou, Shuang Yang, Le Song, and Yuan Qi. Bandit samplers for training graph neural networks. *CoRR*, abs/2006.05806, 2020b.

Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. In *Advances in Neural Information Processing Systems*, pp. 2156–2167, 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. *arXiv preprint arXiv:1901.09342*, 2019c.

Zhewei Wei Ming Chen, Bolin Ding Zengfeng Huang, and Yaliang Li. Simple and deep graph convolutional networks. 2020.

Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5115–5124, 2017a.

Federico Monti, Michael Bronstein, and Xavier Bresson. Geometric matrix completion with recurrent multi-graph neural networks. In *Advances in Neural Information Processing Systems*, pp. 3697–3707, 2017b.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning*, pp. 2014–2023, 2016.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pp. 8026–8037, 2019.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710, 2014.

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 652–660, 2017.

Afshin Rahimi, Trevor Cohn, and Tim Baldwin. Semi-supervised user geolocation via graph convolutional networks. 04 2018.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2020.

Emanuele Rossi, Fabrizio Frasca, Ben Chamberlain, Davide Eynard, Michael Bronstein, and Federico Monti. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198*, 2020.

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, 2008.

Lei Tang and Huan Liu. Relational learning via latent social dimensions. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 817–826. ACM, 2009.

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

Nikil Wale, Ian A Watson, and George Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14(3):347–375, 2008.

Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 38(5): 1–12, 2019.

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

Felix Wu, Amauri H Souza Jr, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q Weinberger. Simplifying graph convolutional networks. In *ICML*, 2019.

Bingbing Xu, Huawei Shen, Qi Cao, Yunqi Qiu, and Xueqi Cheng. Graph wavelet neural network. In *International Conference on Learning Representations*, 2019a.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

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

Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning-Volume 48*, pp. 40–48, 2016.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 974–983. ACM, 2018.

Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *ICLR*, 2016.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In Edwin R. Hancock Richard C. Wilson and William A. P. Smith (eds.), *Proceedings of the British Machine Vision Conference (BMVC)*, pp. 87.1–87.12. BMVA Press, September 2016. ISBN 1-901725-59-6. doi: 10.5244/C.30.87. URL https://dx.doi.org/10.5244/C.30.87.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in neural information processing systems*, pp. 3391–3401, 2017.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graph-SAINT: Graph sampling based inductive learning method. In *International Conference on Learning Representations*, 2020.

Jiani Zhang, Xingjian Shi, Junyuan Xie, Hao Ma, Irwin King, and Dit-Yan Yeung. Gaan: Gated attention networks for learning on large and spatiotemporal graphs. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence*, pp. 339–349, 2018.

Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnns. *International Conference on Learning Representations*, 2020.

Marinka Zitnik and Jure Leskovec. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, 33(14):i190–i198, 2017.

A Exploring Generalized Message Aggregators

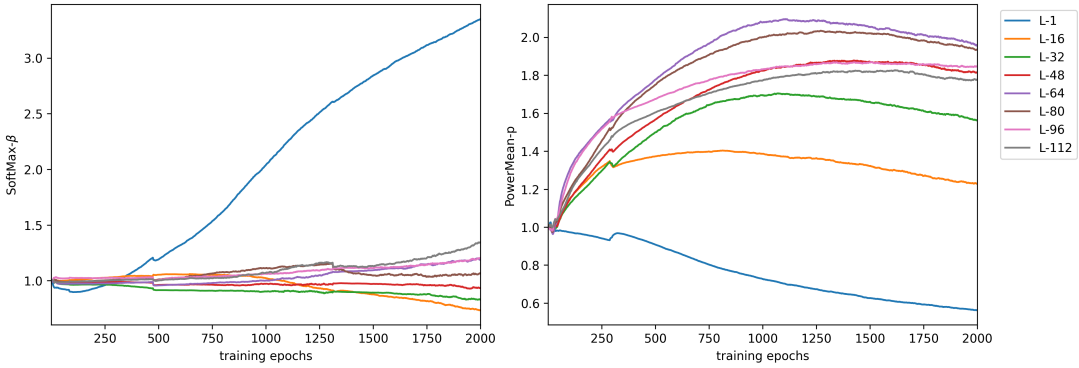
In Table 5 (a) & (b), we examine SoftMax_Agg$_\beta(\cdot)$ and PowerMean_Agg$_p(\cdot)$ aggregators on ogbn-proteins by manually tuning the parameters β and p, respectively. Their performance are measured by test ROC-AUC. Since both are *generalized mean-max aggregations*, they can theoretically perform at least as good as Mean$(\cdot)$ and Max$(\cdot)$ through interpolation. For instance, with 112 layers, SoftMax_Agg$(\cdot)$ performs similarly to Mean$(\cdot)$ (82.20% *vs*. 82.16%) when $\beta = 10^{-3}$. As β increases to 10^2, it achieves slightly better performance than Max$(\cdot)$. Remarkably, 112-layer ResGEN with SoftMax_Agg$(\cdot)$ reaches 86.38% and 86.30% ROC-AUC when $\beta = 10$ and $\beta = 10^4$ respectively. For PowerMean_Agg$(\cdot)$, we find that it reaches almost the same ROC-AUC as Mean when $p = 1$ (arithmetic mean). We also observe that all other orders of mean except $p = 10^{-3}$ (akin to geometric mean) achieve better performance than the arithmetic mean. PowerMean_Agg$(\cdot)$ with $p = 10$ reaches the best ROC-AUC at 86.31% with 112 layers. However, due to some numerical issues in PyTorch (Paszke et al., 2019), we are not able to use larger p. These results empirically validate the discussion on existence of better generalized mean-max aggregators beyond mean and max in Section 4.1.

Table 5: **Exploring the characteristics of Generalized Message Aggregators.** Proposed SoftMax_Agg$(\cdot)$ and PowerMean_Agg$(\cdot)$ can approximate vanilla aggregators (Mean$(\cdot)$ or Max$(\cdot)$) with corresponding parameters. SoftMax_Agg$(\cdot)$ and PowerMean_Agg$(\cdot)$ also go beyond Mean$(\cdot)$ and Max$(\cdot)$ through interpolation and have potentials to achieve better performance. Results reported by measuring test ROC-AUC on the ogbn-protein dataset. Note that due to some numerical issues in PyTorch, we are not able to use larger p for PowerMean_Agg.

(a)	Mean	SoftMax								Max
#Layers	-	10^{-3}	10^{-2}	10^{-1}	1	10	10^2	10^3	10^4	-
3	79.69	79.69	78.90	77.80	81.69	83.24	83.16	83.07	83.21	**83.47**
7	80.84	80.81	80.71	79.83	83.85	83.98	84.66	84.60	**84.68**	84.65
14	82.25	82.44	82.14	81.24	84.39	85.13	84.96	84.99	84.85	**85.16**
28	83.28	83.13	82.47	81.78	85.08	85.07	85.35	85.80	**85.82**	85.26
56	83.52	83.62	83.45	82.86	85.76	85.97	**86.20**	85.98	86.19	86.05
112	83.40	83.50	83.61	83.16	85.77	**86.38**	86.27	86.27	86.30	85.94
avg.	82.16	82.20	81.88	81.11	84.42	84.96	85.10	85.12	**85.17**	85.09
(b)	Mean	PowerMean								Max
#Layers	-	-1	10^{-3}	1	2	3	4	5	10	-
3	79.69	82.34	81.06	78.52	80.23	82.01	81.61	**82.89**	82.89	**83.47**
7	80.84	83.36	81.08	81.02	83.49	83.67	**84.82**	84.54	84.50	84.65
14	82.25	83.73	80.64	82.45	84.15	84.48	84.64	85.00	85.08	**85.16**
28	83.28	84.56	80.92	82.58	84.16	85.20	**85.87**	85.34	85.76	85.26
56	83.52	84.46	80.93	83.49	85.04	85.68	85.90	85.64	85.74	**86.05**
112	83.40	85.13	81.10	83.92	85.47	85.70	86.01	86.09	**86.31**	85.94
avg.	82.16	83.93	80.95	82.00	83.76	84.46	84.81	84.92	85.05	**85.09**

B MORE ANALYSIS ON THE LEARNING DYNAMIC

We have visualized the learning dynamic of SoftMaxSum_Agg$_{(\beta,y)}(\cdot)$ aggregator and PowerMeanSum_Agg$_{(p,y)}(\cdot)$ aggregator in Figure 4. Here we provide analysis on the learning dynamic of the other two proposed generalized aggregators SoftMax_Agg$_{(\beta)}$ and PowerMean_Agg$_{(p)}$ in Figure 5a and Figure 5b, respectively. One can observe that the β of SoftMax_Agg in the 1-st layer learns to increase. Therefore, the aggregator for the first layer behaves like a max aggregation. The large difference between the 1-st layer and other layers shows that our proposed aggregator learns to adapt how to aggregate local information for each layer in the network. From Figure 5b, the p of PowerMean_Agg tends to learn to be larger than 1 except the first layer.



(a) SoftMax_Agg$_{(\beta)}(\cdot)$ (b) PowerMean_Agg$_{(p)}(\cdot)$

Figure 5: **Learning curves of 112-layer DyResGEN with SoftMax and PowerMean**

C DETAILED COMPARISON WITH SOTA

Due to the limit of space, the number of parameters of each model is not included in Table 3. Here, we include them in Table 6. Our proposed model achieves state-of-the-art or on par performance on each dataset with a moderate size of model parameters.

Table 6: **Comparisons with SOTA on seven OGB datasets.** * denotes that virtual nodes are used.

ogbn-proteins	GraphSAGE	GCN	GaAN	GeniePath-BS	Ours (Small-1)	Ours (Small-2)	Ours
test ROC-AUC	77.68 ± 0.20	72.51 ± 0.35	78.03 ± 0.73	78.25 ± 0.35	83.50 ± 0.30	82.69 ± 0.21	**86.16 ± 0.16**
model params.	193,136	96,880	-	316,754	80,446	82,888	2,374,568
ogbn-arxiv	GraphSAGE	GCN	GaAN	GCNII	JKNet	DAGNN	
test acc.	71.49 ± 0.27	71.74 ± 0.29	71.97 ± 0.24	**72.74 ± 0.16**	72.19 ± 0.21	72.09 ± 0.25	72.32 ± 0.27
model params.	218,664	110,120	1,471,506	2,148,648	89,000	43,857	491,176
ogbn-products	GraphSAGE	GCN	ClusterGCN	GraphSAINT	GAT	GraphZoom	
val acc.	78.29 ± 0.16	75.64 ± 0.21	78.97 ± 0.33	80.27 ± 0.26	79.45 ± 0.59	74.06 ± 0.26	**81.64 ± 0.30**
model params.	206,895	103,727	206,895	206,895	751,574	120,251,183	253,743
	GIN	GCN	GIN*	GCN*	HIMP	PNA	
ogbg-molhiv	75.58 ± 1.40	76.06 ± 0.97	77.07 ± 1.49	75.99 ± 1.19	78.80 ± 0.82	**79.05 ± 1.32**	78.87 ± 1.24
model params	1,885,206	527,701	3,336,306	1,978,801	153,029	326,081	531,976
ogbg-molpcba	22.66 ± 0.28	20.20 ± 0.24	27.03 ± 0.23	24.24 ± 0.34		**28.38 ± 0.35**	27.81 ± 0.38*
model params	1,923,433	565,928	3,374,533	2,017,028		6,550,839	5,550,208
ogbg-ppa	68.92 ± 1.00	68.39 ± 0.84	70.37 ± 1.07	68.57 ± 0.61			**77.12 ± 0.71**
model params	1,836,942	479,437	3,288,042	1,930,537			2,336,421
ogbl-collab	GraphSAGE	GCN	DeepWalk				
test hits@50	48.10 ± 0.81	44.75 ± 1.07	50.37 ± 0.34				**52.73 ± 0.47**
model params	460,289	296,449	61,390,187				117,383

D DISCUSSION ON NETWORK DEPTH

Depth & Normalization. In our experiments, we find normalization techniques play a crucial role in training deep GCNs. Without normalization, the training of deep network may suffer from vanishing

gradient or exploding gradient problem. We apply normalization methods such as BatchNorm (Ioffe & Szegedy, 2015) or LayerNorm (Ba et al., 2016) to normalize vertex features. In addition to this, we also propose a *message normalization* (MsgNorm) layer to normalize features on the message level, which can significantly boost the performance of networks with under-performing aggregation functions. The main idea of *MsgNorm* is to normalize the features of the aggregated message $\mathbf{m}_v^{(l)} \in \mathbb{R}^D$ by combining them with other features during the vertex update phase. Suppose we apply the MsgNorm to a simple vertex update function $\text{MLP}(\mathbf{h}_v^{(l)} + \mathbf{m}_v^{(l)})$. The vertex update function becomes as follows:

$$\mathbf{h}_v^{(l+1)} = \boldsymbol{\phi}^{(l)}(\mathbf{h}_v^{(l)}, \mathbf{m}_v^{(l)}) = \text{MLP}(\mathbf{h}_v^{(l)} + s \cdot \|\mathbf{h}_v^{(l)}\|_2 \cdot \frac{\mathbf{m}_v^{(l)}}{\|\mathbf{m}_v^{(l)}\|_2}) \tag{5}$$

where $\text{MLP}(\cdot)$ is a multi-layer perceptron and s is a learnable scaling factor. The aggregated message $\mathbf{m}_v^{(l)}$ is first normalized by its ℓ_2 norm and then scaled by the ℓ_2 norm of $\mathbf{h}_v^{(l)}$ by a factor of s. In practice, we set the scaling factor s to be a learnable scalar with an initialized value of 1. Note that when $s = \|\mathbf{m}_v^{(l)}\|_2 / \|\mathbf{h}_v^{(l)}\|_2$, the vertex update function reduces to the original form. In our experiment, we find *MsgNorm* boosts performance of under-performing aggregation functions such as *mean* and *PowerMean* on ogbn-proteins more than 1%. However, we do not see any significant gain on well-performing aggregation functions such as *SoftMax*, *SoftMaxSum* and *PowerMeanSum*. We leave this for our future investigation.

Depth & Width. In order to gain a larger representational capacity, we can either increase depth or width of networks. In this work, we focus on the depth instead of the width since it is more challenging to train a deeper graph neural network compared to a wider one because of vanishing gradient (Li et al., 2019) and over-smoothing (Li et al., 2018) problems. Deeper neural networks can learn to extract higher-level features. However, given a certain budget of parameters and computation, a well-designed wider networks can be more accurate and efficient than a deep networks. The trade-off of depth and width have already studied in CNNs (Zagoruyko & Komodakis, 2016). We believe that it is also important to study the width of GCNs to reduce the computational overhead.

Depth & Receptive Field & Diameter. There are lots of discussion on whether depth can help for graph neural networks. In our experiments, we find that graph neural networks can gain better performance with proper skip connections, normalization and aggregation functions. A interesting discussion by Rossi et al. (2020) argues that the receptive field of graph neural networks with a few layers can cover the entire graph since most of graph data are 'small-world' graphs with small diameter. Depth may be harmful for graph neural networks. In our experiment, we observe a different phenomenon. For instance, ogbn-proteins dataset with a relatively small diameter as 9 can gain improvement with more than 100 layers. However, what is the optimal depth and for what certain kind of graphs depth help more are still mysteries.

E Proof for Proposition 4

Proof. Suppose we have $N = \big|\mathcal{N}(v)\big|$. We denote the message set as $\mathbf{M} = \{\, \mathbf{m}_1, ..., \mathbf{m}_N \,\}$, $\mathbf{m}_i \in \mathbb{R}^D$. We first show for any message set, $\text{SoftMax\_Agg}_\beta(\mathbf{M}) = \sum_{j=1}^N \frac{\exp(\beta \mathbf{m}_j)}{\sum_{i=1}^N \exp(\beta \mathbf{m}_i)} \cdot \mathbf{m}_j$ satisfies Definition 2 . Let ρ denotes a permutation on the message set $\mathbf{M}$. $\forall \beta \in \mathbb{R}$, for any $\rho \star \mathbf{M} = \{\, \mathbf{m}_{\rho(1)}, ..., \mathbf{m}_{\rho(N)} \,\}$, it is obvious that $\sum_{i=\rho(1)}^{\rho(N)} \exp(\beta \mathbf{m}_i) = \sum_{i=1}^N \exp(\beta \mathbf{m}_i)$ and $\sum_{j=\rho(1)}^{\rho(N)} \exp(\beta \mathbf{m}_j) \cdot \mathbf{m}_j = \sum_{j=1}^N \exp(\beta \mathbf{m}_j) \cdot \mathbf{m}_j$ since the Sum function is a permutation invariant function. Thus, we have $\text{SoftMax\_Agg}_\beta(\mathbf{M}) = \text{SoftMax\_Agg}_\beta(\rho \star \mathbf{M})$. $\text{SoftMax\_Agg}_\beta(\cdot)$ satisfies Definition 2. We now prove $\text{SoftMax\_Agg}_\beta(\cdot)$ satisfies Definition 3, *i.e.* $\lim_{\beta \to 0} \text{SoftMax\_Agg}_\beta(\cdot) = \text{Mean}(\cdot)$ and $\lim_{\beta \to \infty} \text{SoftMax\_Agg}_\beta(\cdot) = \text{Max}(\cdot)$. For the k-th dimension, we have input message features as $\{\, m_1^{(k)}, ..., m_N^{(k)} \,\}$. $\lim_{\beta \to 0} \text{SoftMax\_Agg}_\beta(\{\, m_1^{(k)}, ..., m_N^{(k)} \,\}) = \sum_{j=1}^N \frac{\exp(\beta m_j^{(k)})}{\sum_{i=1}^N \exp(\beta m_i^{(k)})} \cdot m_j^{(k)} = \sum_{j=1}^N \frac{1}{N} \cdot m_j^{(k)} = \frac{1}{N} \sum_{j=1}^N \cdot m_j^{(k)} = \text{Mean}(\{\, m_1^{(k)}, ..., m_N^{(k)} \,\})$. Suppose we have c elements

that are equal to the maximum value m^*. When $\beta \to \infty$, we have:

$$\frac{\exp(\beta m_j^{(k)})}{\sum_{i=1}^{N}\exp(\beta m_i^{(k)})} = \frac{1}{\sum_{i=1}^{N}\exp(\beta(m_i^{(k)} - m_j^{(k)}))} = \begin{cases} 1/c & \text{for } m_j^{(k)} = m^* \\ 0 & \text{for } m_j^{(k)} < m^* \end{cases} \tag{6}$$

We obtain $\lim_{\beta\to\infty}\text{SoftMax\_Agg}_\beta(\{\, m_1^{(k)}, ..., m_N^{(k)} \,\}) = c\cdot\frac{1}{c}\cdot m^* = m^* = \text{Max}(\{\, m_1^{(k)}, ..., m_N^{(k)} \,\})$. It is obvious that the conclusions above generalize to all the dimensions. Therefore, $\text{SoftMax\_Agg}_\beta(\cdot)$ is a generalized mean-max aggregation function. □

F Proof for Proposition 5

Proof. Suppose we have $N = \big|\mathcal{N}(v)\big|$. We denote the message set as $\mathbf{M} = \{\, \mathbf{m}_1, ..., \mathbf{m}_N \,\}$, $\mathbf{m}_i \in \mathbb{R}_+^D$. We have $\text{PowerMean\_Agg}_p(\mathbf{M}) = (\frac{1}{N}\sum_{i=1}^{N}\mathbf{m}_i^p)^{1/p}$, $p \neq 0$. Clearly, for any permutation $\rho \star \mathbf{M} = \{\, \mathbf{m}_{\rho(1)}, ..., \mathbf{m}_{\rho(N)} \,\}$, $\text{PowerMean\_Agg}_p(\rho \star \mathbf{M}) = \text{PowerMean\_Agg}_p(\mathbf{M})$. Hence, $\text{PowerMean\_Agg}_p(\cdot)$ satisfies Definition 2. Then we prove $\text{PowerMean\_Agg}_p(\cdot)$ satisfies Definition 3 *i.e.* $\text{PowerMean\_Agg}_{p=1}(\cdot) = \text{Mean}(\cdot)$ and $\lim_{p\to\infty}\text{PowerMean\_Agg}_p(\cdot) = \text{Max}(\cdot)$. For the k-th dimension, we have input message features as $\{\, m_1^{(k)}, ..., m_N^{(k)} \,\}$. $\text{PowerMean\_Agg}_{p=1}(\{\, m_1^{(k)}, ..., m_N^{(k)} \,\}) = \frac{1}{N}\sum_{i=1}^{N}\cdot m_i^{(k)} = \text{Mean}(\{\, m_1^{(k)}, ..., m_N^{(k)} \,\})$. Assume we have c elements that are equal to the maximum value m^*. When $p \to \infty$, we have:

$$\lim_{p\to\infty}\text{PowerMean\_Agg}_p(\{\, m_1^{(k)}, ..., m_N^{(k)} \,\}) \tag{7}$$

$$= (\frac{1}{N}\sum_{i=1}^{N}(m_i^{(k)})^p)^{1/p} = (\frac{1}{N}(m^*)^p\sum_{i=1}^{N}(\frac{m_i^{(k)}}{m^*})^p)^{1/p} \tag{8}$$

$$= (\frac{c}{N}(m^*)^p)^{1/p} \overset{m^*>0}{=\!=} m^* \tag{9}$$

We have $\lim_{p\to\infty}\text{PowerMean\_Agg}_p(\{\, m_1^{(k)}, ..., m_N^{(k)} \,\}) = m^* = \text{Max}(\{\, m_1^{(k)}, ..., m_N^{(k)} \,\})$. The conclusions above hold for all the dimensions. Thus, $\text{PowerMean\_Agg}_p(\cdot)$ is a generalized mean-max aggregation function. □

G Experimental Details

G.1 Details of Datasets

In this section, we provide the experimental details of the seven OGB datasets.

Node Property Prediction. Three chosen datasets are dealing with protein-protein association networks (ogbn-proteins), paper citation networks (ogbn-arxiv) and co-purchasing network (ogbn-products). Ogbn-proteins is an undirected, weighted, and typed (according to species) graph containing $132,534$ nodes and $39,561,252$ edges. All edges come with 8-dimensional features and each node has an 8-dimensional one-hot feature indicating which species the corresponding protein comes from. Ogbn-arxiv consists of $169,343$ nodes and $1,166,243$ directed edges. Each node is an arxiv paper represented by a 128-dimensional features and each directed edge indicates the citation direction. As an Amazon products co-purchasing network, ogbn-products is an undirected and unweighted graph which is formed by $2,449,029$ nodes and $61,859,140$ edges where nodes are products sold in Amazon that are represented by 100-dimensional features, and edges indicate that the connected nodes are co-purchased. For ogbn-proteins, the prediction task is multi-label and ROC-AUC is used as the evaluation metric. For ogbn-arxiv and ogbn-products, their prediction tasks are both multi-class and evaluated by accuracy.

Graph Property Prediction. Here, we consider three datasets, two of which deals with molecular graphs (ogbg-molhiv and ogbg-molpcba) and the other is biological subgraphs (ogbg-ppa). Ogbg-molhiv has $41,127$ subgraphs and ogbg-molpcba is much bigger which contains $437,929$ subgraphs. For ogbg-ppa, it consists of $158,100$ subgraphs and each subgraph is much denser in comparison to the other two datasets. The tasks of ogbg-molhiv and ogbg-molpcba are both binary classification

while the prediction task of ogbg-ppa is multi-class classification. The former two are evaluated by the ROC-AUC and Average Precision (AP) metric separately. Accuracy is used to assess ogbg-ppa.

Link Property Prediction. We select ogbl-collab, an author collaboration network consisting of $235,868$ nodes and $1,285,465$ edges for link prediction task. Each node in the graph comes with a 128-dimensional feature vector representing an author and edges indicate the collaboration between authors. The task is to predict the future author collaboration relationships given the past collaborations. Each true collaboration is ranked among a set of $100,000$ randomly-sampled negative collaborations, and the ratio of positive edges that are ranked at K-place or above (*Hits@k*, *k* is 50 here) is counted as the evaluation metric.

G.2 Details of Results and Implementation

For a fair comparison with SOTA methods, we provide results on each dataset by averaging the results from 10 independent runs. We provide the details of the model configuration on each dataset. All models are implemented based on PyTorch Geometric (Fey & Lenssen, 2019) and all experiments are performed on a single NVIDIA V100 32GB.

ogbn-proteins. For both ogbn-proteins and ogbg-ppa, there is no node feature provided. We initialize the features of nodes through aggregating the features of their connected edges by a Sum aggregation, *i.e.* $\mathbf{x_i} = \sum_{j \in \mathcal{N}(i)} \mathbf{e}_{i,j}$, where $\mathbf{x_i}$ denotes the initialized node features and $\mathbf{e}_{i,j}$ denotes the input edge features. We train a 112-layer DyResGEN with SoftMax_Agg$_\beta(\cdot)$ aggregator. A hidden channel size of 64 is used. A layer normalization and a dropout with a rate of 0.1 are used for each layer. We train the model for 2000 epochs with an Adam optimizer with a learning rate of 0.001.

ogbn-arxiv. We train a 28-layer ResGEN model with SoftMax_Agg$_\beta(\cdot)$ aggregator where β is fixed as 0.1. We convert this directed graph into undirected and add self-loop. Full batch training and test are applied. A batch normalization is used for each layer. The hidden channel size is 128. We apply a dropout with a rate of 0.5 for each layer. An Adam optimizer with a learning rate of 0.001 is used to train the model for 2000 epochs.

ogbn-products. A 14-layer ResGEN model with SoftMax_Agg$_\beta(\cdot)$ aggregator where β is fixed as 0.1 is trained for ogbn-products with self-loop added. We apply mini-batch training scenario by randomly partitioning the graph into 10 subgraphs and do full-batch test. For each layer, a batch normalization is used. The hidden channel size is 128. We apply a dropout with a rate of 0.5 for each layer. An Adam optimizer with a learning rate of 0.001 is used to train the model for 1000 epochs.

ogbg-molhiv. We train a 7-layer DyResGEN model with SoftMax_Agg$_\beta(\cdot)$ aggregator where β is learnable. A batch normalization is used for each layer. We set the hidden channel size as 256. A dropout with a rate of 0.2 is used for each layer. An Adam optimizer with a learning rate of 0.0001 are used to train the model for 300 epochs.

ogbg-molpcba. A 14-layer ResGEN model with SoftMax_Agg$_\beta(\cdot)$ aggregator where β is fixed as 0.1 is trained. In addition, the original model performs message passing over augmented graphs with virtual nodes added. A batch normalization is used for each layer. We set the hidden channel size as 256. A dropout with a rate of 0.5 is used for each layer. An Adam optimizer with a learning rate of 0.01 are used to train the model for 300 epochs.

ogbg-ppa. We initialize the node features via a Sum aggregation. We train a 28-layer ResGEN model with SoftMax_Agg$_\beta(\cdot)$ aggregator where β is fixed as 0.01. We apply a layer normalization for each layer. The hidden channel size is set as 128. A dropout with a rate of 0.5 is used for each layer. We use an Adam optimizer with a learning rate of 0.01 to train the model for 200 epochs.

ogbl-collab. The whole model used to train on link prediction task consists of two parts: a 7-layer DyResGEN model with SoftMax_Agg$_\beta(\cdot)$ aggregator where β is learnable and a 3-layer link predictor model. A batch normalization is used for each layer in DyResGEN model. We set the hidden channel size as 128. An Adam optimizer with a learning rate of 0.001 are used to train the model for 400 epochs.