EEGNN: Edge Enhanced Graph Neural Network with a Bayesian Nonparametric Graph Model

Yirui Liu[1,2] **Xinghao Qiao**[1] **Liying Wang**[3] **Jessica Lam**[4]
[1]London School of Economics and Political Science
[2]J.P. Morgan
[3]Bayes Business School, City, University of London
[4]University of Oxford

Abstract

Training deep graph neural networks (GNNs) poses a challenging task, as the performance of GNNs may suffer from the number of hidden message-passing layers. The literature has focused on the proposals of over-smoothing and under-reaching to explain the performance deterioration of deep GNNs. In this paper, we propose a new explanation for such deteriorated performance phenomenon, mis-simplification, that is, mistakenly simplifying graphs by preventing self-loops and forcing edges to be unweighted. We show that such simplifying can reduce the potential of message-passing layers to capture the structural information of graphs. In view of this, we propose a new framework, edge enhanced graph neural network (EEGNN). EEGNN uses the structural information extracted from the proposed Dirichlet mixture Poisson graph model (DMPGM), a Bayesian nonparametric model for graphs, to improve the performance of various deep message-passing GNNs. We propose a Markov chain Monte Carlo inference framework for DMPGM. Experiments over different datasets show that our method achieves considerable performance increase compared to baselines.

1 INTRODUCTION

Graph neural networks (GNNs) (Zhou et al., 2020; Wu et al., 2020) are important tools for analyzing graph data, such as social network (You et al., 2020), transportation network (Chen et al., 2021a), molecular graph (Huang et al., 2020), biological network (Zhang et al., 2021), financial transaction network (Wang et al., 2021), academic citation graph (Xu et al., 2021), and knowledge graph (Ji et al., 2021). GNNs have become popular with their state-of-the-art performance by applying deep learning methodologies to graphs. Among them, message passing neural networks (MPNN) (Gilmer et al., 2017) uses message-passing layers to compute node embeddings. Examples of MPNNs include graph convolutional neural networks (GCN) (Kipf and Welling, 2017), GraphSAGE (Hamilton et al., 2017), graph attention networks (GAT) (Veličković et al., 2018), and gated graph neural networks (GGNN) (Li et al., 2016). Similar to standard multi-layer perceptron (MLP) in deep learning, the message passing layer in a GNN framework aggregates information from the local neighbors of each node, and then transforms the information via an activation function into the embedding (Hamilton, 2020). A node embedding can aggregate information over N hop neighbors, in the form of N hidden message-passing layers, thus incorporating further reaches of the graph.

Although deeper layers in non-graph neural networks often achieve better performance (Krizhevsky et al., 2012; He et al., 2016), GNNs typically perform best with only 2 to 4 hop neighbors, that is, 2 to 4 hidden layers. In contrast, using a larger number of layers, termed as deep stacking, may lead to a substantial drop in the performance for GNNs (Klicpera et al., 2019; Rong et al., 2019; Li et al., 2020; Chen et al., 2020b). One explanation for this phenomenon is the *over-smoothing*. By applying graph convolution repeatedly over many hidden layers, the representation of the nodes will be indistinguishable. As a result, the *over-smoothing* can jeopardize the performance of deep GNNs. Another explanation is the *under-reaching*. When GNNs aggregate messages over long paths, the information propagation across distant nodes in the graph becomes difficult because it is susceptible to bottlenecks (Alon and Yahav, 2020). This causes GNNs to perform poorly in predicting tasks that require remote interaction (Singh et al., 2021; Hwang et al., 2021).

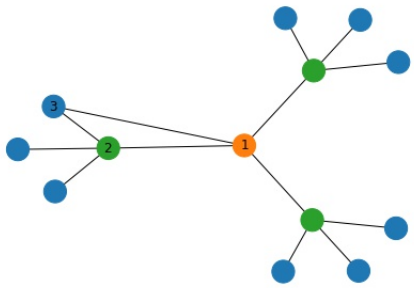
Many efforts have been devoted to addressing these limitations. To handle the over-smoothing, DropEdge (Rong et al., 2019) and DropNode (Huang et al., 2021) were proposed to randomly remove a certain number of edges or nodes from

Proceedings of the 26th International Conference on Artificial Intelligence and Statistics (AISTATS) 2023, Valencia, Spain. PMLR: Volume 206.

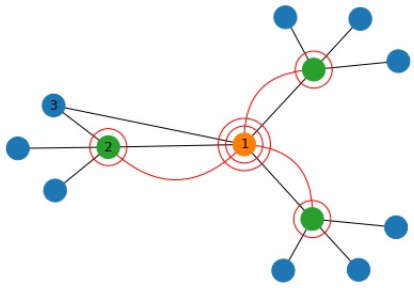
the input graph at each training epoch. These methods are likened to Dropout (Srivastava et al., 2014), which randomly drops hidden neurons in neural networks to prevent overfitting. On the contrary, to address the under-reaching, virtual edges (Gilmer et al., 2017), super nodes (Scarselli et al., 2009; Hwang et al., 2021), or short-cut edges (Allamanis et al., 2018) can be added to the original graph. However, none of the aforementioned methods consider adding or removing based on the structural information of the graph. Instead, the pattern of deciding which nodes or edges to be added or removed comes from an arbitrarily random selection. Although dropout has been effective in non-graph neural networks, its random removal and addition of nodes can disturb the graph structure, thus compromising the performance of GNNs that relies on the structure to propagate information.

Different from the *over-smoothing* and *under-reaching*, we propose a new explanation for performance deterioration of deeper GNNs from the perspective of misusing edge structural information, *mis-simplification*, explained as follows. Most observed graphs are recorded as simple graphs, where self-loops are not allowed, and all edges are unweighted and undirected (Shafie, 2015). In a natural way, GNNs are designed for learning such simple graphs that can be constructed by collapsing multiple edges into a single edge as well as removing self-loops. This approach, however, discards the information inherent in the original network. Take one example, for a source node connected to many neighboring target nodes (see node 1 in Figure 1a), its self-loop has an equal weight to neighboring non-loop edge, which may under-weigh the importance of this node. Take another example, no matter how similar the two nodes are (see nodes 1 and 2 in Figure 1a), only one edge is allowed to connect the pair of nodes, and as a result, the information passing between both nodes is restricted. Furthermore, edge $(1, 2)$ should play a more important role in message passing than edge $(1, 3)$, because node 2 is a key node with 3 sub-nodes in total, while node 3 is just a sub-node of node 2. However, typical GNNs treat these two edges indifferently as they are equally weighted in the simple graph. Therefore, such *mis-simplification* can reduce the potential of message-passing layers to capture structural information in GNNs.

To solve this issue, we propose an edge-enhanced graph neural network (EEGNN), which incorporates edge structural information in the message-passing layer. First, we assume that there is an underlying *virtual multigraph*, allowing for self-loops and for multiple edges between pairs of nodes, and the observed graph model can be viewed as a transformation of the virtual multigraph. As illustrated in Figure 1, the above Figure 1ais the original observed simple graph, while the below Figure 1b is the corresponding virtual graph. Second, to build the virtual multigraph that can capture the edge structural information, we propose the Dirichlet mixture Poisson graph model, a Bayesian non-



(a) The observed simple graph



(b) The corresponding virtual multigraph

Figure 1: The observed simple graph versus the virtual multigraph. The red edges are virtual edges in the virtual multigraph. In particular, the red circles are virtual self-loops.

parametric model. Following Caron and Fox (2017), the interactions between nodes are modelled by assigning a *sociability* parameter to each node. Then, the counts of edges are generated from a Poisson distribution, where the Poisson rate is the product of sociability parameters of the nodes in two ends. Finally, in the framework of EEGNN, we can then replace the observed graph in a GNN with the virtual multigraph. In this architecture, message-passing layers can then assign weights proportionally to the importance of the edges, thus passing the information from nodes to nodes in a more reasonable manner.

The main contribution of our paper is fourfold.

- We outline a new explanation for the poor performance of deep GNNs;
- We propose a new way to enhance existing GNN methods by utilizing the structural information of graphs;
- We propose a Bayesian nonparametric graph model and its Monte Carlo Markov chain (MCMC) inference procedure;
- We demonstrate the superior sample performance of our proposal over existing methods through the experiments on six real datasets and a financial application.

2 Preliminaries

2.1 GNN and Message Passing Layer

We begin by introducing some notation. Let $G = (V, E)$ be a graph with node set $V = \{v_1, \ldots, v_{|V|}\}$ and edge set $E = \{e_1, \ldots, e_{|E|}\}$, where $|V|$ and $|E|$ denote the number of nodes and edges in G, respectively. The adjacency matrix $A \in \mathbb{R}^{|V|\times|V|}$ is defined as $A_{ij} = 1$ if $(v_i, v_j) \in E$ and 0 otherwise. The corresponding degree matrix $D \in \mathbb{R}^{|V|\times|V|}$ is defined as $D = \text{diag}(D_1, \ldots, D_{|V|})$, where $D_i = \sum_{j=1}^{|V|} A_{ij}$. We denote the data matrix by $X \in R^{m\times|V|}$, whose j-th column corresponds to a m-dimensional feature vector of node j.

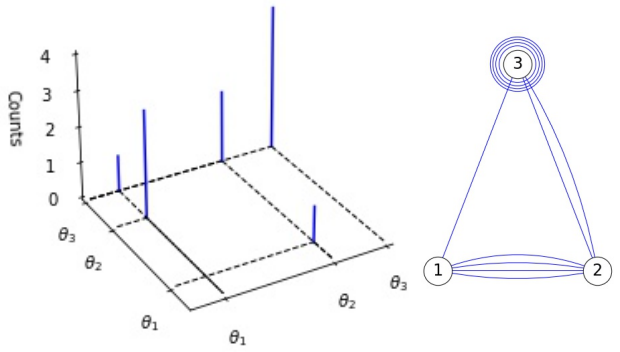
GNN is a neural network model to process graphs for node classification, edge prediction and graph classification (Gori et al., 2005; Zhou et al., 2020; Wang et al., 2021). Within various GNNs, information is exchanged between nodes and is updated by neural networks via message passing layers (Gilmer et al., 2017). Specifically, the initial representation, h_i^0 for node i, is generated by a function of this node's features. Then, the message passing layers update the representation based on this node's neighbors. The message passing contains two steps: the aggregation step and the update step. Denote the representation for node i in layer l by h_i^l. A message passing layer in GNN can be expressed as

$$h_i^{l+1} = \text{UPDATE}(h_i^l, \text{AGGREGATE}(h_j^l \mid j \in \mathcal{N}_i)), \quad (1)$$

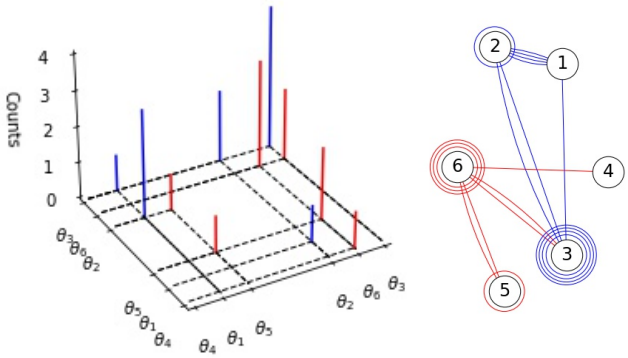
where AGGREGATE$(\cdot)$ denotes a permutation-invariant function, such as the sum, mean, and maximum, to send information from one node to another through edges, and UPDATE$(\cdot)$ denotes linear or nonlinear differentiable functions such as MLP, $\mathcal{N}_i$ denotes the neighborhood of node i, that is, the set of nodes directly connected to node i. For example, the vanilla GCN uses $h_i^{l+1} = \sigma(\sum_{j=1}^{|V|} \tilde{P}_{ij} h_j^{l+1} W^l \mid j \in \mathcal{N}_i \cup \{i\})$, or in matrix form, $H^{l+1} = \sigma(\tilde{P} H^l W^l)$ (Kipf and Welling, 2017), APPNP uses $H^{l+1} = (1-\alpha)\tilde{P}H^l + \alpha H^0$ (Klicpera et al., 2019), and GCNII uses $H^{l+1} = \sigma\big((1-\alpha)\tilde{P}H^l + \alpha H^0)((1-\beta)I_n + \beta W^l)\big)$ (Chen et al., 2020b), where $\tilde{P} = (D+I)^{-\frac{1}{2}}(A+I)(D+I)^{-\frac{1}{2}}$, H^l is the representation for all nodes in layer l, σ and W^l are respectively the activation function and the corresponding weight in a neural network layer, and α and β are hyperparameters. The formulas above show that the GNN treats each edge with equal weight and hence leads to *mis-simplification*. In order to solve this issue, we adopt a Bayesian nonparametric sparse graph model to generate the virtual edges and virtual multigraph.

2.2 Bayesian Nonparametric Sparse Graph Model

In contrast with other graph models that are based on node feature embeddings, Caron and Fox (2017) represent the observed graph G as a point process on $\mathbb{R}^2$,



(a) Graph model in Caron and Fox (2017)



(b) Dirichlet mixture Poisson graph model (DMPGM)

Figure 2: Bayesian nonparametric graph model. Figure 2a illustrates the model in Caron and Fox (2017), while Figure 2b illustrates the proposed graph model in this paper. The left sub-figures show the proxy for nodes, θ_is, and the number of edges among them. The right sub-figures display the corresponding multigraph. In Figure 2b, red and blue are used to indicate two clusters in the edges. The circles around nodes denote self-loops. Finally, the number of circles or links denotes the multiplicity.

$G = \sum_{i,j} z_{i,j} \delta_{(\theta_i,\theta_j)}$, where Dirac function $\delta_{(\theta_i,\theta_j)}$ is equal to 1 at (θ_i, θ_j) and equal to 0 elsewhere, $z_{i,j}$ is the multiplicity for edge (i, j), and θ_i is a proxy for node i on the real axis, as illustrated in Figure 2a. Note that the same definition for node i is also applied to node j, but we omit the explanation for node j to avoid redundancy. This representation specifies the source and target nodes for each edge. To model the possibility for two nodes constructing an edge, a sociability parameter $w_i > 0$ is assigned to node i for each $i = 1, \ldots, |V|$. Following Aldous (1997), the graph model can be factorized as $p(A_{ij} = 1) = 1 - \exp(-2w_i w_j)$ for $i \neq j$ and $p(A_{ii} = 1) = 1 - \exp(-w_i^2)$ otherwise. This is equivalent to modelling an unobserved integer-valued multigraph as $z_{ij} \sim \text{Poisson}(w_i w_j)$ and setting $A_{ij} = \mathbb{1}_{z_{ij}+z_{ji}>0}$. To model the sparsity property in real graphs, that is, $|E| = o\big(|V|^2\big)$, the sociability is generated from a completely random measure with infinite activity

(Caron and Fox, 2017), such as gamma process, stable process and inverse Gaussian process (Ghosal and Van der Vaart, 2017). See also Appendix A for a short review of completely random measures. This model allows for self-loop and multi-edges, and thus can be used to build a virtual multigraph. However, as the Poisson intensity is factorized as the outer product of a vector with itself, only one feature for each node is considered, which restricts the capability of this model in confronting real data. To address this issue, we propose a novel model in Section 3.1 below.

3 METHODOLOGY

3.1 Dirichlet Mixture Poisson Graph Model

To adopt the latent community information among nodes in the graph, mixed-membership stochastic block model (Airoldi et al., 2008) associates each node with latent cluster distributions. In an analogy, we add cluster-membership features to each pair of edges instead of nodes. Specifically, we extend the graph model in Caron and Fox (2017) by proposing the following Dirichlet mixture Poisson graph model (DMPGM),

$$\begin{aligned}
\pi = (\pi_1, \pi_2, \cdots) &\sim \text{GEM}(\alpha), \\
W_0 = \sum_{i=1}^{\infty} w_{0,i}\delta_{\theta_i} &\sim \text{CRM}(\kappa, v), \\
W_k = \sum_{i=1}^{\infty} w_{k,i}\delta_{\theta_i} &\sim \Gamma\text{P}(W_0), \\
z_{ij} \sim \text{Poisson}(\sum_{k=1}^{\infty} \pi_k w_{k,i} \times w_{k,j}&), \\
A_{ij} = \min(z_{ij} + z_{ji}, 1)\mathbb{1}_{i\neq j}&,
\end{aligned} \tag{2}$$

for $i, j, k \in \mathbb{N}^+$, where GEM(α) is the distribution for atom sizes of a Dirichlet process DP(α) and each atom corresponds to a distinct cluster. Moreover, CRM(κ, v) denotes a completely random measure with $v^c(dw, d\theta) = \kappa(d\theta)v(dw)$ as its Levy measure, and $\Gamma\text{P}(H)$ denotes gamma process with the base measure H. See Appendix A for details of these stochastic processes. We summarize the probabilistic generative steps as follows. First, the cluster distribution π is assigned with a prior GEM(α), which allows for infinitely many clusters. Second, we use a hierarchical structure to generate values for the node sociability parameter in each cluster. W_0, sampled from a completely random measure, is used as the base measure in $\Gamma\text{P}(W_0)$ for W_k such that $w_{k,i}$ belongs to gamma distribution parameterized by $w_{0,i}$, $w_{k,i} \sim \text{Gamma}(w_{0,i})$. This hierarchical setting is designed to ensure the components in W_k share atom locations (Teh et al., 2006; Liu et al., 2022). Finally, following Caron and Fox (2017), an undirected multigraph $\sum_{i,j} z_{ij}\delta(\theta_i, \theta_j)$ is generated from a Poisson process, where z_{ij} is the Poisson-distributed multiplicity for edge (i, j). By aggregating multiple edges to a single edge for each pair of nodes and removing self-loops, a simple graph is transformed from the multigraph. The corresponding adjacent matrix $A = (A_{ij})$ to the observable simple graph can then be generated. An example of DMPGM is illustrated in Figure 2b.

DMPGM can be equivalently expressed under a mixture model framework. Specifically, a set of edges in each cluster is sampled from Poisson($\pi_k \bar{w}_k^2$), where $\bar{w}_k = \sum_{i=1}^{\infty} w_{k,i}$. As a consequence, this is equivalent to sampling the total number of edges n from Poisson(λ) with $\lambda = \sum_{k=1}^{\infty} \pi_k \bar{w}_k^2$, and then assigning each edge a cluster membership from Categorical$\big(\frac{\pi_1\bar{w}_1^2}{\lambda}, \frac{\pi_2\bar{w}_2^2}{\lambda}, \dots\big)$. Following the same methodology, for each edge, a pair of nodes is then sampled from Categorical$(\frac{w_{k,1}}{\bar{w}_k}, \frac{w_{k,2}}{\bar{w}_k}, \dots)$ in the cluster k. Hence, a relationship between edges and nodes is constructed. We summarise this equivalent expression for DMPGM as follows,

$$\begin{aligned}
n &\sim \text{Poisson}(\sum_{k=1}^{\infty} \pi_k \bar{w}_k^2), \\
k &\sim \text{Categorical}\big(\frac{\pi_1\bar{w}_1^2}{\lambda}, \frac{\pi_2\bar{w}_2^2}{\lambda}, \dots\big), \\
i, j &\sim \text{Categorical}\big(\frac{w_{k,1}}{\bar{w}_k}, \frac{w_{k,2}}{\bar{w}_k}, \dots\big),
\end{aligned} \tag{3}$$

where other structures in equation (2) remain the same. In Appendix C, we show that DMPGM enjoys similar properties as the model in Caron and Fox (2017) in the following theorem.

Theorem 1 *The graph constructed by DMPGM is sparse if CRM in (2) has infinite activity.*

For example, using the gamma process as the completely random measure leads to a sparse graph in the DMPGM, which makes it more effective for modeling real-life data.

It is worth noting that DMPGM extends the model in Caron and Fox (2017) by assuming that edges can belong to different clusters. As a result, DMPGM is more flexible and applicable in modelling real data. We also note that DMPGM is distinct from the overlapping communities graph model (Todeschini et al., 2020) and graph Poisson factorization (Zhou, 2015), because we assign a Dirichlet prior for the clustering distribution, and hence can allow a nonparametric estimation of the number of edge clusters. Moreover, Williamson (2016) uses the hierarchical Dirichlet process (HDP) (Teh et al., 2006) to construct the graph. However, as HDP only models the node distribution within a cluster, the number of edges is ignored. As a result, this model cannot be used for EEGNN framework. Finally, a generative model that shares some fundamental similarities with DMPGM is proposed by Ricci et al. (2022). However, this concurrent work does not investigate the use of a Bayesian nonparametric graph model to improve GNNs.

3.2 MCMC Inference Framework

We propose a detailed MCMC framework to infer the posteriors for DMPGM in a nonparametric way. Following Caron and Fox (2017) and Liu et al. (2022), the posterior distribution for $W_k = \sum_{i=1}^{\infty} w_{k,i}\delta_{\theta_i}$, $k \geqslant 0$, are restricted to the weights $\{w_{k,i}\}$ because the locations $\{\theta_i\}$ of both observed and unobserved nodes are not likelihood identifiable, thus being ignored. Moreover, given the observed nodes set V, the weights for each W_k are truncated to a $(|V|+1)$-dimensional vector, $\boldsymbol{w}_k = (w_{k,0}, w_{k,1}, \ldots, w_{k,|V|})^{\mathrm{T}}$, where $w_{k,i}$ corresponds to the weight on an observed node i for $1 \leqslant i \leqslant |V|$, and $w_{k,0}$ is the sum of weights for all unobserved nodes. Similarly, the posterior distribution for π is truncated to a $(K+1)$-dimensional vector, $\boldsymbol{\pi} = (\pi_0, \pi_1, \ldots, \pi_K)^{\mathrm{T}}$, where K is the truncated number of clusters and is inferred adaptively in Step 4 below, and π_0 corresponds to the cluster without any observation. Consequently, given the truncation levels $|V|$ and K for W_0 and π, respectively, DMPGM contains the following parameters to infer: $\boldsymbol{\pi}$, $\{\boldsymbol{w}_k\}_{k\geqslant 0}$, $\boldsymbol{z} = \{z_{ij}\}_{A_{ij}=1}$ and cluster membership $\boldsymbol{c} = \{c_{ijl}\}_{A_{ij}=1, 1\leqslant l\leqslant z_{ij}}$.

We next propose a MCMC inference framework that can infer the number of edge clusters in a Bayesian nonparametric manner in the following steps.

Step 1 Update $w_{0,1}, \ldots, w_{0,|V|}|\bar{w}_0, \boldsymbol{z}, \boldsymbol{c}$ using Hamiltonian Monte Carlo (Kroese et al., 2011), where $\bar{w}_0 = \sum_{i=0}^{|V|} w_{0,i}$ with the log-posterior and its gradient provided in Appendix B.1.

Step 2 Update $w_{k,1}, \ldots, w_{k,|V|}|\bar{w}_k, w_0, \boldsymbol{z}, \boldsymbol{c}$ for $k = 1, \ldots, K$ given the conjugacy, where $\bar{w}_k = \sum_{i=0}^{|V|} w_{k,i}$. We sample $\widetilde{w}_{k,i} \sim \text{Dirichlet}(\nu_0, \nu_1, \ldots, \nu_{|V|})$, where $\nu_i = w_{0,i} + \sum_{j=1}^{|V|} n_{k,i}$, $n_{k,i} = \sum_{j=1}^{|V|}\sum_{l=1}^{z_{ij}}\{\mathbb{1}_{c_{ijl}=k} + \mathbb{1}_{c_{jil}=k}\}$, and then compute $w_{k,i} = \bar{w}_k\widetilde{w}_{k,i}$.

Step 3 Update $\pi_0, \pi_1, \ldots, \pi_K|\boldsymbol{z}, \boldsymbol{c}$ using the conjugacy. Analogous to Step 2, we sample $\pi_k \sim \text{Dirichlet}(n_0, n_1, \ldots, n_k)$, where $n_k = \sum_{i=1}^{|V|}\sum_{j=1}^{|V|}\sum_{l=1}^{z_{ij}} \mathbb{1}_{c_{ijl}=k}$ for $k > 0$ and $n_0 = \alpha$.

Step 4 Update the latent edge cluster membership $c_{ijl}|\{\boldsymbol{w}_k\}_{k\geqslant 0}, \boldsymbol{\pi}$ for each pair (i,j) such that $A_{ij} = 1$ and for $l = 1, \ldots, z_{ij}$. For each edge we sample from the multinomial distribution $p(c_{ijl} = k) \propto \pi_k w_{k,i} w_{k,j}$ for $k = 0, 1 \ldots, K$. In this step, if $k = 0$ is sampled, we add a new cluster (Teh et al., 2006; Bryant and Sudderth, 2012; Liu et al., 2022), and increase the truncated number of clusters from K to $K+1$.

Step 5 Update the unobserved $z_{ij} \mid \boldsymbol{\pi}, \boldsymbol{w}_k \sim$ Truncated–Poisson$(\sum_{k=0}^{K} \pi_k w_{k,i} w_{k,j})$ for each pair (i,j) such that $A_{ij} = 1$, where truncated Poisson is a conditional probability distribution of a Poisson-distributed random variable with strictly positive counts (Cohen, 1960).

Step 6 Update the $\bar{w}_k$ and $\bar{w}_0$ using Metropolis–Hastings (Kroese et al., 2011) algorithm based on the log-posterior provided in Appendix B.2.

We iterate over Steps 1–6 until convergence. For the MCMC algorithm, the global variables are updated in linear time, and the Monte Carlo step iteratively samples from K clusters. Therefore, the computational complexity is dominated by $O\big(K \max\{|V|, |E|\}\big)$.

3.3 Edge Enhanced Message Passing

In conventional message passing layers built from a simple graph, information for node i is obtained from edges connected to its neighboring nodes in $\mathcal{N}_i$ and from its self-loop. In these layers, each edge (i,j) for $j \in \mathcal{N}_i \cup \{i\}$ has equal weight, resulting in *mis-simplification* of the more complex structural information for the GNN, as described in Section 1. To overcome this *mis-simplification*, we sample artificial edges given the estimated DMPGM, from which we construct a virtual multigraph

$$G^* = (V, E, r), \;\; r\big((i,j)\big) = z_{ij}, \tag{4}$$

where the multiplicity-map $r : E \to \mathbb{N}^+$ assigns to each edge an integer to represent its multiplicity, and z_{ij} in DMPGM is defined in (2) and is inferred from Step 5 in Section 3.2. In this way, we can extract the edge structural information, via the inferred multiplicity for each edge, using the DMPGM model to build the virtual multigraph. For example, as illustrated in Figure 1, two artificial self-loops are added to nodes 1, one artificial self-loop is added to nodes 2, and the edge $(1,2)$ is assigned with multiplicity 2, where the multiplicity is determined by $z_{11} = 2$, $z_{22} = 1$ and $z_{12} + z_{21} = 2$, respectively. We then replace the original simple graph in the message passing layers by the generated virtual multigraph, that is,

$$\begin{aligned} h_i^{l+1} =& \text{UPDATE}^l(h_i^l, r(i,i), \\ & \text{AGGREGATE}^l(h_j^l, r(i,j) \mid j \in \mathcal{N}_i)). \end{aligned} \tag{5}$$

For example, for GCN, APPNP and GCNII, we replace $\tilde{P}$ by $\hat{P}$, where $\hat{P}$ is defined as

$$\begin{aligned} \hat{P} =& \hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}, \;\; \hat{A} = (\hat{A}_{ij} = z_{ij}), \\ & \hat{D} = \text{diag}(\hat{D}_i = \sum_j z_{ij}). \end{aligned} \tag{6}$$

In addition, as the virtual multigraph already contains self-loops, there is no need to add the self-loops again to the message passing layers. This is different from conventional GNNs, where the self-loops are often added and forced to be a single edge. Though GIN (Xu et al., 2019) and

JKNet (Xu et al., 2018) also assign different weights for self-loops empirically, we are the first to propose a method to systematically determine the relative weights for self-loops and other edges.

In summary, conditional on the updated parameters of DMPGM in each iteration, we sample the multiplicity of each edge. Then a set of multiedges and self-loops are generated from DMPGM, which can be used to build a virtual graph and update GNN trainable parameters. We present the proposed EEGNN algorithm in Algorithm 1.

Algorithm 1: EEGNN Algorithm

Iterate Step 1 to Step 6 in Section 3.2 till the MCMC chains converge.
Set up initialization of trainable parameters in EEGNN.
repeat
1. Build the virtual graph and sample $\hat{P}$ according to (6),
2. Use $\hat{P}$ to replace $\tilde{P}$,
3. Update GNN parameters using the gradient descent,
4. Obtain a new sampling for the parameters in DMPGM by implementing Step 1 to Step 6,

until the convergence of the loss function of EEGNN

It is worth noting that we opted not to employ the stochastic block model in our study as it produces only dense graphs where the number of edges increases proportionally to the square of the number of nodes, whereas real-world networks tend to be sparse (Caron and Fox, 2017). Moreover, the stochastic block model does not allow for constructing a virtual multi-graph on edges. To build the virtual multi-graph, it is needed to use a statistical model on edges instead of on nodes.

3.4 Comparison with Other Methods

Our proposed method, EEGNN, addresses the performance deterioration of deep GNNs by using the structural information extracted from a Bayesian nonparametric graph model, DMPGM, to improve the performance of various deep message-passing GNNs. This is in contrast to relevant methods such as the attention and edge-label guided GNNs (Zhou et al., 2022) and edge-enhanced graph convolution networks (Cui et al., 2020), which focus on integrating syntactic dependency or dependency label information into GCN to perform event detection or named entity recognition, respectively. Edge-feature-enhanced GNNs (Gong and Cheng, 2019), another competing method, focuses on integrating edge features instead of extracting edge information based on the observed graph in an unsupervised-learning fashion. Our EEGNN framework differs from these methods as it addresses the issue of *mis-simplification* in deep GNNs and uses structural information from DMPGM to improve performance.

4 EXPERIMENTS

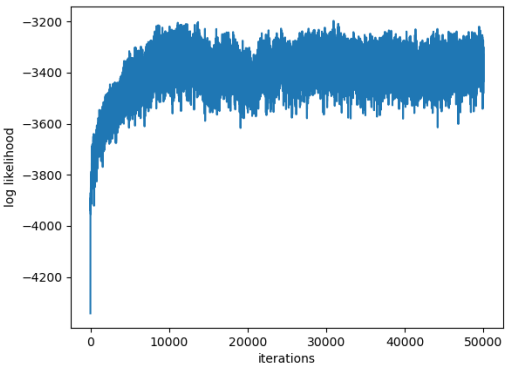
4.1 Datasets and Bayesian Estimation

In this section, we demonstrate through real data examples that EEGNN can effectively use the edge structural information to improve the performance for various GNNs. We conduct empirical experiments to compare EEGNN with representative baselines across six well-established network datasets. First, *Cora*, *Citeseer*, and *PubMed* are standard benchmark datasets for citation networks (Yang et al., 2016). In these networks, nodes represent papers, and edges indicate cross citations between papers. Node features are the bag-of-words embedding of the contents, and node labels are academic subjects. Second, *Texas*, *Cornell*, and *Wisconsin* are webpage cross-link networks (Pei et al., 2020). Their nodes represent web pages of universities, and edges represent hyperlinks between them. Node features are bag-of-words embedding of the websites. Node labels contain five categories for the webs including students, projects, courses, staff, and faculty. Statistics for these datasets are summarized in in Table 1.

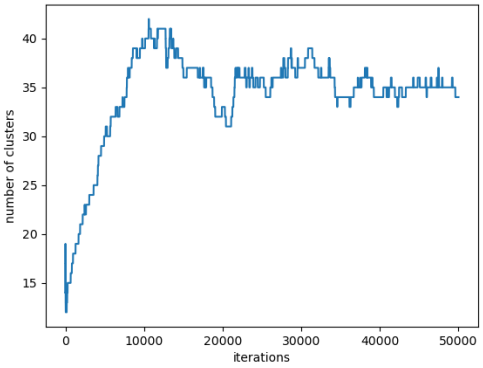
Table 1: Graph datasets statistics.

Dataset	*Cora*	*Citeseer*	*PubMed*	*Texas*	*Wisconsin*	*Cornell*
Nodes	2,708	3,327	19,717	183	183	183
Edges	5,429	4,732	44,338	309	499	295
Degrees	3.88	2.84	4.50	3.38	5.45	3.22
Features	1,433	3,703	500	1,703	1,703	1,703
Classes	7	6	3	5	5	5

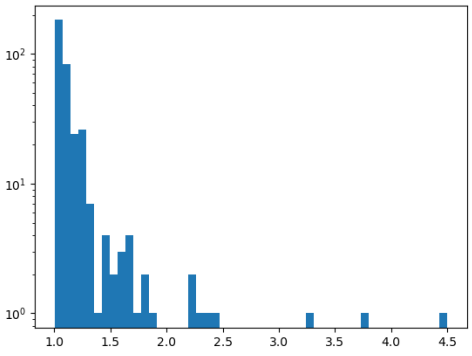
Our experiments are implemented by using a gamma process as the completely random measure in (2). Following Section 3.2, we infer the parameters of DMPGM using MCMC in the following way. We use population based training (Jaderberg et al., 2017) to tune the hyperparameters in DMPGM. For each dataset, we grow the MCMC chain up to 50,000 epochs. Figures 3a and 3b display the log-likelihood and number of clusters with respect to training epochs for the *Texas* dataset. (The training results for the other datasets are shown in Appendix D.) Figure 3a shows that the log-likelihood of *Texas* for the DMPGM converges after 10,000 epochs. Moreover, benefiting from the Bayesian nonparametric model, we can infer the number of edge clusters in a data-adaptive manner (Liu et al., 2022). Figure 3b shows that the inferred number of edges per node (termed as *multiplicity* of virtual edges per node) rises from an initial value of 10 to 50 at the start of training and then converges to around 35. The inferred edge multiplicity is displayed in the histograms in Figure 3c. These histograms show that a large proportion of the edges in the observed graph have underlying multi-edges, suggesting the *mis-simplification* in the original observed graph.



(a) The training log likelihood of DMPGM



(b) The inferred number of edge clusters



(c) Histogram of the expected multiplicity of virtual edges

Figure 3: The MCMC inference results for *Texas*.

4.2 Comparison with Baselines

With the inference results of DMPGM, following Section 3.3, we implement the experiments to compare baseline GNNs and their edge enhanced versions. For the baseline GNNs, we chose SGC (Wu et al., 2019) and its variant, including APPNP (Klicpera et al., 2019) and GCNII (Chen et al., 2020a), and hence name their edge enhanced versions as EE-SGC, EE-APPNP and EE-GCNII, respectively. To make a fair comparison, we follow the settings of the 'sweet point' GNN hyperparameter configuration in Chen et al. (2021b) for all datasets. The details of these hyperparameter settings are collected in Appendix E. For all experiments, the GNNS are trained with a maximum of 1000 epochs and an early stopping patience of 100 epochs. To analyze the effect of EEGNN with different numbers of layers, we run the experiments for 2, 16, 32 and 64 layers. We randomly split node features in each dataset into training and test sets, train the baseline GNNs and the edge enhanced versions using the same training set, and then compute the node clustering accuracy on the test set. In the transductive learning framework for GNNs, it is noted that the edge information is not partitioned as described in Kipf and Welling (2017). We repeat this procedure 50 times for each model and dataset. The mean predictive accuracy and the corresponding standard deviation are reported in Table 2.

We observe a few apparent patterns. First, EEGNN can improve the performance of the baseline models in most cases. For example, SGC, the backbone GNN for various models, performs poorly with 32 layers (see Table 2c). However, with the aid of our EEGNN framework, the accuracy of the SGC model is increased by more than 6% for *Cora*, and by approximately 2% across other candidate datasets. Moreover, SGC performs even worse with 64 layers for *Cora* and *Pubmed* (see Table 2d). EEGNNs largely improve the prediction accuracy in both cases, by 9.89% and 23.30%, respectively. It is worth noting that the improvements are attained without changing any other settings. As using virtual multigraph or observed simple graph brings in the only difference, this provides strong evidence to reveal that EEGNN can be used as a tool to enhance baseline GNNs by alleviating the *mis-simplification* problem.

Second, for APPNP and GCNII, EEGNNs achieve similar accuracies on the *Cora*, *Citeseer* and *PubMed* datasets, but substantially improve the performance on *Texas*, *Wisconsin* and *Cornell*. Especially, with 64 layers, EE-GCNII for *Texas* leads to more than 6% improvement, and EE-APPNP for *Citeseer* results in more than 10% increase in the predictive accuracy. On the other hand, as APPNP and GCNII have already reached relatively high accuracy (approximately $70\% - 80\%$) on the *Cora*, *Citeseer* and *PubMed*, further enhancement to higher accuracy tends to be difficult.

Finally, we observe that EEGNN has a larger impact on the performance of deeper SGC on the *Cora*, *Citeseer* and *PubMed*. With only 2 layers, edge enhanced versions behave slightly worse than baseline models. However, with 32 or 64 layers, EEGNNs achieve considerable improvements. This is because the *mis-simplification* applies to all layers. Therefore, the distortion of edge structural information is accumulated from the first to the last layer, resulting in severe performance deterioration.

Table 2: Results on real datasets: mean accuracy (%) ± standard deviation (%)

(a) Number of layers: 2

	Cora	*Citeseer*	*PubMed*	*Texas*	*Wisconsin*	*Cornell*
SGC	**77.01±0.34**	69.18±0.35	75.46±0.28	56.16±4.99	48.59±6.59	57.84±2.76
EE-SGC	76.78±0.29	**69.60±0.37**	**75.80±0.21**	**61.24±6.48**	**53.45±8.95**	**58.92±3.55**
APPNP	**82.22±0.39**	**71.73±0.76**	**79.41±0.48**	61.41±5.27	52.55±7.44	57.73±2.74
EE-APPNP	81.48±0.47	71.45±0.54	78.90±0.52	**66.80±3.74**	**66.23±2.93**	**60.17±6.00**
GCNII	**82.21±0.67**	67.65±0.96	**77.91±1.71**	61.35±8.18	**72.51±4.91**	74.22±8.75
EE-GCNII	81.94±0.51	**81.48±0.59**	77.30±0.97	**64.22±9.02**	70.94±6.10	**75.68±9.78**

(b) Number of layers: 16

	Cora	*Citeseer*	*PubMed*	*Texas*	*Wisconsin*	*Cornell*
SGC	**73.11±0.43**	67.79±0.56	70.45±0.17	56.27±4.92	48.63±6.62	57.84±2.76
EE-SGC	73.07±0.34	**68.55±0.35**	**70.60±0.25**	**59.96±6.46**	**50.59±8.72**	**57.84±2.76**
APPNP	**83.70±0.20**	72.51±0.52	**80.42±0.30**	60.76±5.05	53.29±7.09	57.68±2.79
EE-APPNP	83.47±0.66	**73.20±0.92**	77.90±0.36	**66.08±4.62**	**66.08±3.17**	**61.30±7.18**
GCNII	**84.77±0.37**	72.30±0.80	78.60±0.52	66.38±8.69	70.71±2.40	74.49±8.98
EE-GCNII	84.10±0.57	**72.50±1.40**	**78.81±0.66**	**73.30±3.85**	**78.94±4.90**	**75.24±8.08**

(c) Number of layers: 32

	Cora	*Citeseer*	*PubMed*	*Texas*	*Wisconsin*	*Cornell*
SGC	59.94±0.56	66.17±0.50	68.97±0.19	56.16±4.99	48.59±6.59	57.84±2.76
EE-SGC	**66.46±0.83**	**67.68±0.44**	**70.68±0.68**	**59.46±6.16**	**50.59±8.72**	**58.92±3.15**
APPNP	83.55±0.50	72.11±0.64	80.22±0.34	61.57±5.28	52.71±7.34	57.78±2.75
EE-APPNP	**83.79±0.39**	**72.47±0.53**	79.23±0.27	**66.00±4.33**	**66.39±3.09**	**61.84±7.56**
GCNII	85.34±0.53	73.26±0.86	**79.89±0.33**	70.49±5.48	69.06±2.70	74.05±8.56
EE-GCNII	**85.70±0.41**	**73.45±1.40**	79.72±0.43	**75.24±3.72**	**79.37±0.43**	**74.86±7.84**

(d) Number of layers: 64

	Cora	*Citeseer*	*PubMed*	*Texas*	*Wisconsin*	*Cornell*
SGC	25.65±1.93	63.08±0.52	40.98±1.73	56.16±4.99	48.55±6.58	57.84±2.76
EE-SGC	**35.54±1.36**	**65.42±0.17**	**64.28±0.82**	**59.46±6.16**	**50.31±8.42**	**58.92±3.15**
APPNP	83.58±0.49	72.10±0.48	**80.42±0.42**	61.19±5.29	53.06±7.10	57.68±2.74
EE-APPNP	**83.76±0.41**	**72.16±0.65**	79.94±0.22	**66.00±4.08**	**66.63±3.12**	**61.75±7.43**
GCNII	85.46±0.31	**73.44±1.00**	**80.08±0.37**	69.57±5.70	68.63±1.05	73.19±8.83
EE-GCNII	**85.54±0.59**	72.24±1.26	79.93±0.46	**75.62±3.65**	**76.57±3.89**	**73.26±7.39**

4.3 Application in Financial Data

GNN is widely used in the financial industry for the prediction of stock and bond prices (Wang et al., 2021; Sharma and Sharma, 2020; Feng et al., 2022). To evaluate the efficacy of EEGNN in real-world financial data, we conduct an empirical study using EE-SGC to replace SGC in the current literature and then make a comparison. We use the component stocks from the 'FTSE UK 50 index' with high capitalization and complete records between 2016-01-01 and 2017-12-31. We first construct the graph based on the Pearson correlations between stock returns, by connecting two stocks if their correlation is larger than 0.3. As shown in Figure 4, stocks, indicated by nodes are connected according to their pairwise correlation. Then, we build a learning pipeline using a sequential model of a long short-term memory (LSTM) network, SGC/EE-SGC, and a fully-connected layer. The model was trained using the data in 2016 and tested on the data in 2017. Moreover, the historical returns were used as input data, and the mean squared error between the model outputs and the realized next-day returns was used as the loss function. The Long 20% strategy is adopted to build the portfolio as described in Pacreau et al. (2021). For each trading day, we build a long only portfolio consisting of the top 20% stocks according to the predictive returns. The accumulated returns of the portfolio are shown in Figure 5, where the initial portfolio value is set to be \$100. The results show that the portfolio constructed using EEGNN, which achieved better predictive accuracy, had higher returns.

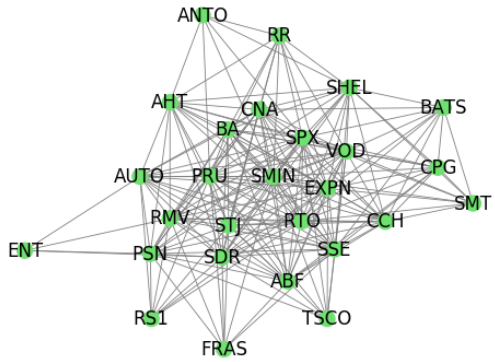


Figure 4: The graph between FTSE UK 50 component stocks. Nodes in green denote individual stocks with their abbreviations in capital letters.

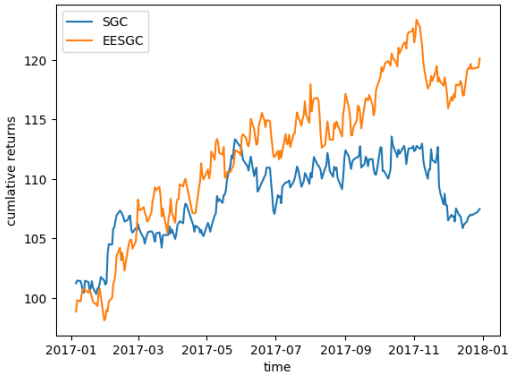


Figure 5: The comparison of cumulative returns using SGC and EE-SGC.

5 CONCLUSION

This paper presents a novel explanation for the performance deterioration of deeper GNNs: *mis-simplification*. We propose DMPGM, a Bayesian nonparametric graph model and its MCMC inference framework. Using the information obtained from DMPGM, we replace the original simple graph by the virtual graph, and use the virtual graph to aggregate the information in the graph. The experiments over various real datasets demonstrate that EEGNN can improve the performance of baseline GNN methods. Our paper paves a new way to use information extracted by statistical graph modelling to improve the performance of GNNs. One limitation of our proposal is that EEGNN only adds the virtual edges to the observed graph without removing edges according to the structural information. It is left for future work to develop a framework that allows to add and remove edges with the structural information simultaneously.

Acknowledgments

We thank the anonymous reviewers for their useful comments during the review process.

References

Airoldi, E. M., Blei, D. M., Fienberg, S. E., and Xing, E. P. (2008). Mixed membership stochastic blockmodels. *Journal of Machine Learning Research*, 9(65):1981–2014.

Aldous, D. (1997). Brownian excursions, critical random graphs and the multiplicative coalescent. *The Annals of Probability*, 25(2):812–854.

Allamanis, M., Brockschmidt, M., and Khademi, M. (2018). Learning to represent pograms with graphs. In *International Conference on Learning Representations*.

Alon, U. and Yahav, E. (2020). On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations*.

Bryant, M. and Sudderth, E. B. (2012). Truly nonparametric online variational inference for hierarchical Dirichlet processes. In *Advances in Neural Information Processing Systems 25*, pages 2699–2707.

Caron, F. and Fox, E. B. (2017). Sparse graphs using exchangeable random measures. *Journal of the Royal Statistical Society: Series B*, 79(5):1295–1366.

Chen, B., Bécigneul, G., Ganea, O.-E., Barzilay, R., and Jaakkola, T. (2021a). Optimal transport graph neural networks. *arXiv:2006.04804*.

Chen, D., Lin, Y., Li, W., Li, P., Zhou, J., and Sun, X. (2020a). Measuring and relieving the over-smoothing

problem for graph neural networks from the topological view. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3438–3445.

Chen, M., Wei, Z., Huang, Z., Ding, B., and Li, Y. (2020b). Simple and deep graph convolutional networks. In *Proceedings of the 37th International Conference on Machine Learning*, pages 1725–1735. PMLR.

Chen, T., Zhou, K., Duan, K., Zheng, W., Wang, P., Hu, X., and Wang, Z. (2021b). Bag of tricks for training deeper graph neural networks: a comprehensive benchmark study. *arXiv:2108.10521*.

Cohen, A. C. (1960). Estimating the parameter in a conditional poisson distribution. *Biometrics*, 16(2):203.

Cui, S., Yu, B., Liu, T., Zhang, Z., Wang, X., and Shi, J. (2020). Edge-enhanced graph convolution networks for event detection with syntactic relation. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2329–2339.

Feng, S., Xu, C., Zuo, Y., Chen, G., Lin, F., and XiaHou, J. (2022). Relation-aware Dynamic Attributed Graph Attention Network for Stocks Recommendation. *Pattern Recognition*, 121:108119.

Ferguson, T. S. (1973). A Bayesian analysis of some nonparametric problems. *The Annals of Statistics*, 1(2):209–230.

Ghosal, S. and Van der Vaart, A. (2017). *Fundamentals of Nonparametric Bayesian Inference*. Cambridge University Press, Cambridge.

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.

Gong, L. and Cheng, Q. (2019). Exploiting edge features for graph neural networks. In *Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition*, pages 9211–9219.

Gori, M., Monfardini, G., and Scarselli, F. (2005). A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734.

Hamilton, W., Ying, Z., and Leskovec, J. (2017). Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, volume 30.

Hamilton, W. L. (2020). Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 14(3):1–159.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778.

Huang, K., Xiao, C., Glass, L. M., Zitnik, M., and Sun, J. (2020). SkipGNN: Predicting molecular interactions with skip-graph networks. *Scientific Reports*, 10(1):1–16.

Huang, W., Rong, Y., Xu, T., Sun, F., and Huang, J. (2021). Tackling over-smoothing for general graph convolutional networks. *arXiv:2008.09864*.

Hwang, E., Thost, V., Dasgupta, S. S., and Ma, T. (2021). Revisiting virtual nodes in graph neural networks for link prediction.

Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., and others (2017). Population based training of neural networks. *arXiv:1711.09846*.

Ji, S., Pan, S., Cambria, E., Marttinen, P., and Philip, S. Y. (2021). A survey on knowledge graphs: Representation, acquisition, and applications. *IEEE Transactions on Neural Networks and Learning Systems*, 33(2):494–514.

Kingman, J. F. C. (1993). *Poisson Processes*. Clarendon Press, Oxford.

Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*.

Klicpera, J., Bojchevski, A., and Günnemann, S. (2019). Predict then propagate: Graph neural networks meet personalized PageRank. In *International Conference on Learning Representations*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 25.

Kroese, D. P., Taimre, T., and Botev, Z. I. (2011). *Handbook of Monte Carlo Methods*. Wiley.

Li, G., Xiong, C., Thabet, A., and Ghanem, B. (2020). DeeperGCN: All you need to train deeper GCNs. *arXiv:2006.07739*.

Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. S. (2016). Gated graph sequence neural networks. In *International Conference on Learning Representations*.

Liu, Y., Qiao, X., and Lam, J. (2022). CATVI: Conditional and adaptively truncated variational inference for hierarchical bayesian nonparametric models. In *Proceedings of the 25th International Conference on Artificial Intelligence and Statistics*, pages 3647–3662.

Pacreau, G., Lezmi, E., and Xu, J. (2021). Graph neural networks for asset management. *SSRN Scholarly Paper*.

Pei, H., Wei, B., Chang, K. C.-C., Lei, Y., and Yang, B. (2020). Geom-gcn: Geometric graph convolutional networks. In *International Conference on Learning Representations*.

Ricci, F. Z., Guindani, M., and Sudderth, E. B. (2022). Thinned random measures for sparse graphs with overlapping communities. In *Advances In Neural Information Processing systems*.

Rong, Y., Huang, W., Xu, T., and Huang, J. (2019). DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. In *International Conference on Learning Representations*.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2009). The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80.

Shafie, T. (2015). A multigraph approach to social network analysis. *Journal of Social Structure*, 16(1):1–21.

Sharma, S. and Sharma, R. (2020). Forecasting Transactional Amount in Bitcoin Network Using Temporal GNN Approach. In *International Conference on Advances in Social Networks Analysis and Mining*, pages 478–485.

Singh, A., Huang, Q., Huang, S. L., Bhalerao, O., He, H., Lim, S.-N., and Benson, A. R. (2021). Edge proposal sets for link prediction. *arXiv:2106.15810*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

Teh, Y. W., Jordan, M. I., Beal, M. J., and Blei, D. M. (2006). Hierarchical Dirichlet processes. *Journal of the American Statistical Association*, 101(476):1566–1581.

Todeschini, A., Miscouridou, X., and Caron, F. (2020). Exchangeable random measures for sparse and modular graphs with overlapping communities. *Journal of the Royal Statistical Society: Series B*, 82(2):487–520.

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

Wang, J., Zhang, S., Xiao, Y., and Song, R. (2021). A Review on graph neural network methods in financial applications. *arXiv:2111.15367*.

Williamson, S. A. (2016). Nonparametric network models for link prediction. *Journal of Machine Learning Research*, 17(202):1–21.

Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. (2019). Simplifying graph convolutional networks. In *International Conference on Machine Learning*, pages 6861–6871.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Philip, S. Y. (2020). A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24.

Xu, F., Yao, Q., Hui, P., and Li, Y. (2021). Automorphic wquivalence-aware graph neural network. In *Advances in Neural Information Processing Systems*, volume 34.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2019). How powerful are graph neural networks? In *International Conference on Learning Representations*.

Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.-i., and Jegelka, S. (2018). Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, pages 5453–5462.

Yang, Z., Cohen, W., and Salakhudinov, R. (2016). Revisiting semi-supervised learning with graph embeddings. In *International conference on Machine Learning*, pages 40–48. PMLR.

You, Y., Chen, T., Sui, Y., Chen, T., Wang, Z., and Shen, Y. (2020). Graph contrastive learning with augmentations. In *Advances in Neural Information Processing Systems*, volume 33, pages 5812–5823.

Zhang, W., Sheng, Z., Jiang, Y., Xia, Y., Gao, J., Yang, Z., and Cui, B. (2021). Evaluating deep graph neural networks. *arXiv:2108.00955*.

Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., and Sun, M. (2020). Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81.

Zhou, M. (2015). Infinite edge partition models for overlapping community detection and link prediction. In *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics*, pages 1135–1143.

Zhou, R., Xie, Z., Wan, J., Zhang, J., Liao, Y., and Liu, Q. (2022). Attention and edge-label guided graph convolutional networks for named entity recognition. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 6499–6510.

Supplementary Material for 'EEGNN: Edge Enhanced Graph Neural Network with a Bayesian Nonparametric Graph Model'

This supplementary material contains a short review of completely random measures in Appendix A, the details of MCMC steps in Appendix B, technical proofs and derivations in Appendix C, the results of MCMC for several datasets in Appendix D, hyperparameter settings in Appendix E, and computational complexity analysis and code in Appendix F.

A Completely Random Measure

Completely random measures (Ghosal and Van der Vaart, 2017), including gamma process, inverse Gaussian process and stable process, are commonly used as priors for infinite-dimensional latent variables in Bayesian nonparametric models, because their realizations are atomic measures with countable-dimensional supports. Suppose that $(\Omega, \mathscr{F})$ is a Polish sample space, Θ is the set of all bounded measures on $(\Omega, \mathscr{F})$ and $\mathscr{M}$ is a σ-algebra on Θ. A complete random measure from $(\Theta, \mathscr{M})$ into $(\Omega, \mathscr{F})$ can be characterized by its Laplace transform (Kingman, 1993),

$$\mathrm{E}\big[e^{-tP(A)}\big] = \exp\Big\{-\int_A \int_{(0,\infty]} (1-e^{-t\pi}) v^c(dx, ds)\Big\},$$

where A is any measurable subset of Ω and $v^c(dx, ds)$ is called the intensity measure. If $v^c(dx, ds) = \kappa(dx)v(ds)$, where $\kappa(\cdot)$ and $v(\cdot)$ are measures on Ω and $(0, \infty]$, respectively, the completely random measure is homogeneous and $v(\cdot)$ is called the Lévy measure. If $\int_0^\infty v(ds) = \infty$, the complete random measure is finite activity.

We can view this completely random measure as a Poisson process on the product space $\Omega \times (0, \infty]$ using the intensity measure and denote this completely random measure as CRM(κ, v). For example, the gamma process $\Gamma\mathrm{P}(\kappa)$ has Lévy measure $v(ds) = s^{-1}e^{-s}ds$ such that $Q(A) \sim \Gamma(\kappa(A), 1)$ if $Q \sim \Gamma\mathrm{P}(\kappa)$, where $\Gamma(\alpha, \beta)$ is a gamma distribution with density $\frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1}e^{-\beta x}$. Therefore, its normalization, Dirichlet process $P \sim \mathrm{DP}(\kappa)$ (Ferguson, 1973) satisfies

$$\big(P(A_1), \ldots, P(A_n)\big) \sim \mathrm{Dirichlet}\big(\kappa(A_1), \ldots, \kappa(A_n)\big)$$

for any partition $\Omega = (A_1, \ldots, A_n)$, where $\bigcup_{i=1}^n A_i = \Omega$ and $A_i \bigcap A_j = \varnothing$ for any i and j. Griffiths–Engen–McCloskey (GEM) distribution, which is the distribution of the weights in a Dirichlet process. For $(\pi_1, \pi_2, \cdots) \sim \mathrm{GEM}(\alpha)$, it can be sampled by $\pi_i = g_i \prod_{l=1}^{i-1} g_l$, where $g_i \sim \mathrm{Beta}(1, \alpha)$ independently (Ghosal and Van der Vaart, 2017).

B MCMC technical details and derivations

B.1 Derivations for Step 1

With the setup of DMPGM in (2) and the formula of moments for Dirichlet-multinomial distribution, we obtain that

$$p(w_{0,1}, \ldots, w_{0,|V|} \mid \bar{w}_0, \boldsymbol{z}, \boldsymbol{c}) \propto \prod_{k=1}^K \frac{\Gamma(\bar{w}_0)}{\Gamma(\bar{w}_0 + n_k)} \prod_{i=0}^N \frac{\Gamma(w_{0,i} + n_{k,i})}{\Gamma(w_{0,i})} \cdot \prod_{i=1}^N v(w_{0,i}) \cdot u\big(\bar{w}_0 - \sum_{i=1}^{|V|} w_{0,i}\big),$$

where $n_{k,i} = \sum_{j=1}^{|V|} \sum_{l=1}^{z_{ij}} \{\mathbb{1}_{c_{ijl}=k} + \mathbb{1}_{c_{jil}=k}\}$, $v(\cdot)$ is the weight intensity measure for the complete random measure of W_0, and $u(\cdot)$ is the density function for $W_0(\Omega)$ that can be derived using its Laplace transform. To infer the posterior distributions for these parameters, we present the gradient of the log-posterior with respect to w_0, which will be used in Hamiltonian Monte Carlo,

$$\begin{aligned}\nabla_{w_{0,i}} \log p(w_{0,1}, \ldots, w_{0,|V|} \mid \bar{w}_0, \boldsymbol{z}, \boldsymbol{c}) = &\sum_{i=1}^{|V|} \sum_{k=1}^K \big\{\Phi(n_{k,i} + w_{0,i}) - \Phi(w_{0,i})\big\} \\ &+ \sum_{i=1}^{|V|} \nabla_{w_{0,i}} \log v(w_{0,i}) + \nabla_{w_{0,i}} \log u(\bar{w}_0 - \sum_{i=1}^{|V|} w_{0,i}),\end{aligned}$$

where Φ is the digamma function.

B.2 Derivations for Step 6

By the formulas of the densities for Poisson distribution and gamma distribution, we have that

$$p(\bar{w_k} \mid \bar{w}_0, \boldsymbol{\pi}, \boldsymbol{c}, \boldsymbol{z}) \propto \frac{(\pi_k \bar{w}_k^2)^{n_k} e^{-\pi_k \bar{w}_k^2}}{n_k!} \cdot \frac{1}{\Gamma(\bar{w}_0)} \bar{w}_k^{\bar{w}_0 - 1} e^{-\bar{w}_k}.$$

Therefore, the log-posterior with respect to $\bar{w}_k$ is

$$\log p(\bar{w}_k \mid \bar{w}_0, \boldsymbol{\pi}, \boldsymbol{c}, \boldsymbol{z}) = (2n_k + \bar{w}_0 - 1) \log \bar{w}_k - \bar{w}_k - \bar{w}_k^2 \pi_k + \text{constant}.$$

Similarly, following Caron and Fox (2017) and Liu et al. (2022), we obtain that

$$p(\bar{w}_0 \mid \bar{w}_k, \boldsymbol{\pi}, \boldsymbol{c}, \boldsymbol{z}) \propto \prod_{k=1}^{K} \frac{1}{\Gamma(\bar{w}_0)} \bar{w}_k^{\bar{w}_0 - 1} e^{-\bar{w}_k} \cdot u(\bar{w}_0).$$

and hence the corresponding log-posterior is

$$\log p(\bar{w}_0 \mid \bar{w}_k, \boldsymbol{\pi}, \boldsymbol{c}, \boldsymbol{z}) = \log u(\bar{w}_0) + \bar{w}_0 \sum_{k=1}^{K} \log(\bar{w}_k) - K \log \Gamma(\bar{w}_0) + \text{constant}.$$

C Technical Proofs and Derivations

C.1 Proof of Theorem 1

The proof for Theorems 3 and 4 in Caron and Fox (2017) can be directly adapted to DMPGM, Therefore, we only provide a sketch of the proof. First, we show that Theorem 3 in Caron and Fox (2017) also holds for DMPGM. We use

$$\tilde{D}_{ij} \mid \{W_k\} \sim \text{Poisson}\big(\sum_k \pi_k W_k([i-1, i]) W([j-1, j])\big),$$

to replace (54) in Appendix C.2 of Caron and Fox (2017). Consequently, (55) holds because for any k we have

$$W_k([0, \alpha])/W_0([0, \alpha]) = O(1) \text{ almost surely as } \alpha \to \infty. \tag{B.1}$$

Second, we show that Theorem 4 in Caron and Fox (2017) also holds for DMPGM. Specifically, (59) becomes

$$X_n \mid \{W_k^{(2)}\} \sim \text{Poisson}\Big[\frac{1}{2}\psi\big\{W(\mathcal{S}_n^{(2)})\big\}\Big],$$

so that (62) in (Caron and Fox, 2017) can be achieved by (B.1). Finally, we complete the proof of Theorem 1 for DMPGM by keeping the remaining parts of the proof of Theorem 4 in Caron and Fox (2017) unchanged.

D Inference results for DMPGM

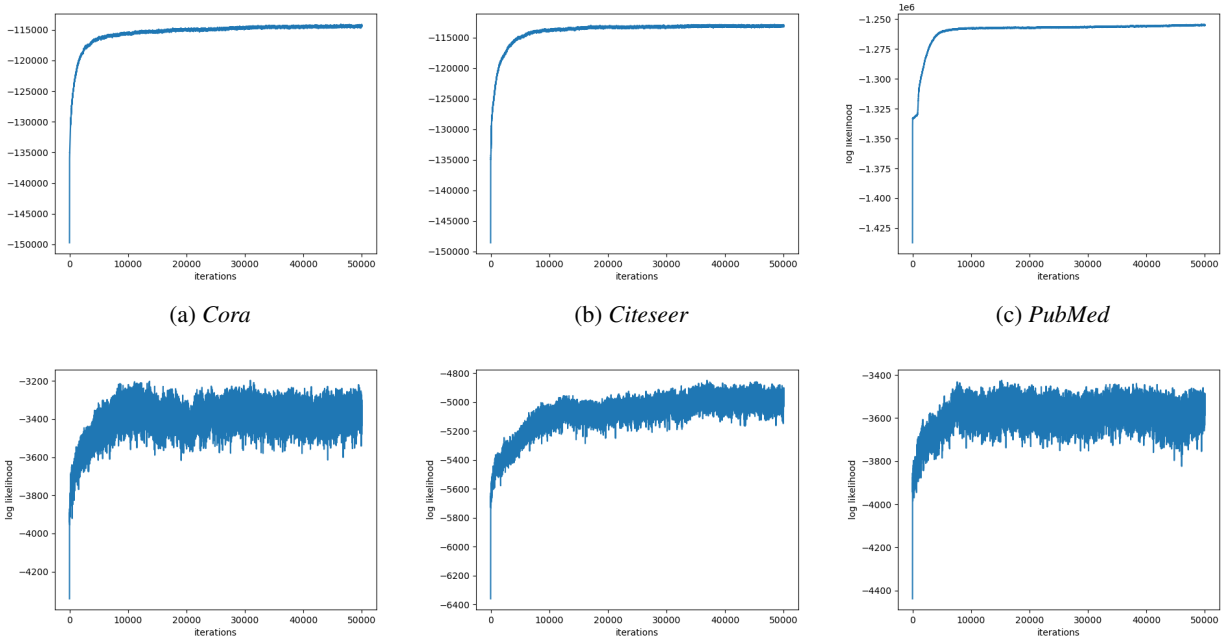
The log likelihood and the number of edge clusters in the training process are shown in Figure 6 and Figure 7, respectively. The inferred edge multiplicity is shown in Figure 8.

E Hyperparameters

We list the hyperparameters used in our experiments in Table 3 below.

F Data and Code

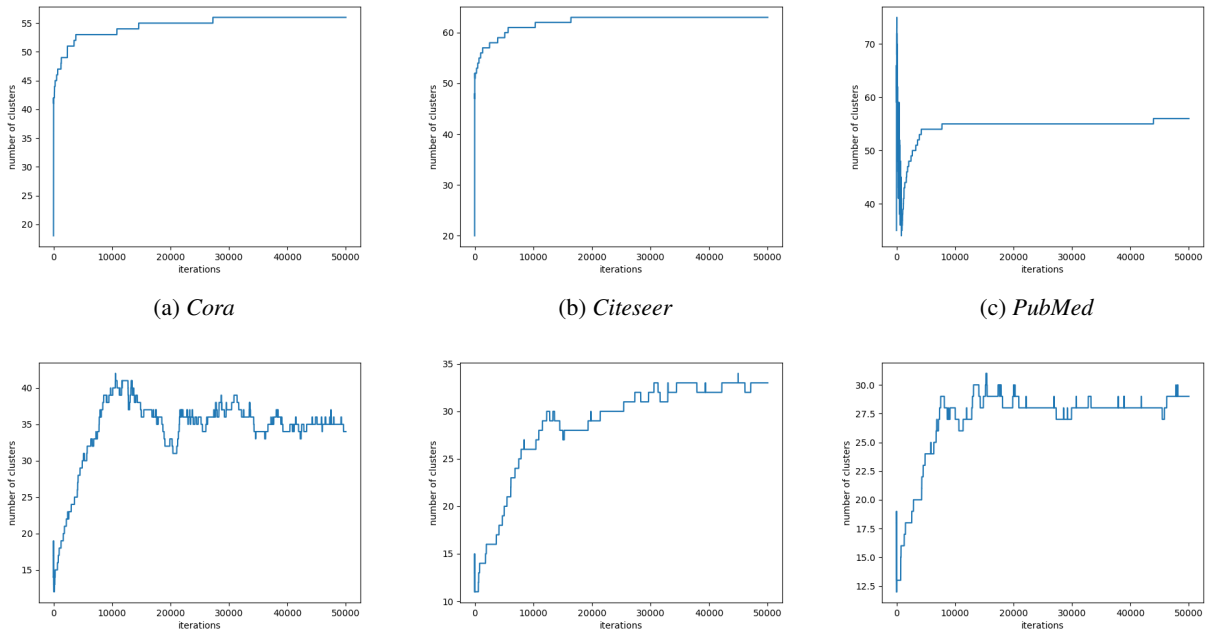
We obtained the datasets from the publically available source `https://pytorch-geometric.readthedocs.io/en/latest/modules/datasets.html`. All data do not contain personally identifiable information or offensive content. We conducted our experiments on a c5d.4xlarge instance on the AWS EC2 platform, with 16 vCPUs and 32 GB RAM. The codes for training conventional GNNs are from `https://github.com/VITA-Group/Deep_GCN_Benchmarking` under MIT license.



(a) *Cora* (b) *Citeseer* (c) *PubMed*

(d) *Texas* (e) *Wisconsin* (f) *Cornell*

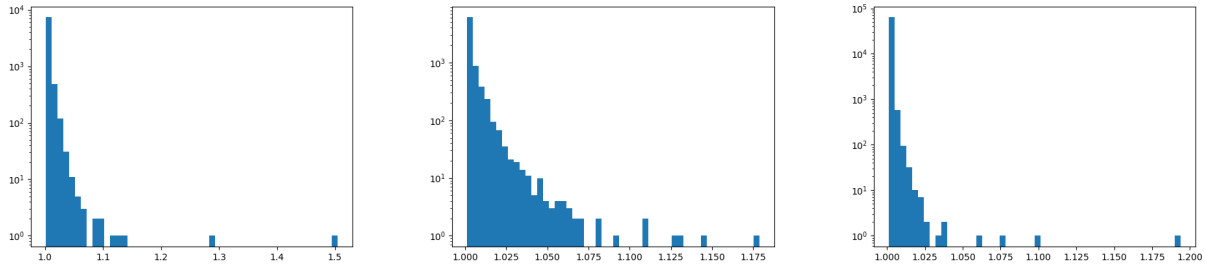
Figure 6: Log-likelihood over the course of the MCMC chain for each dataset.



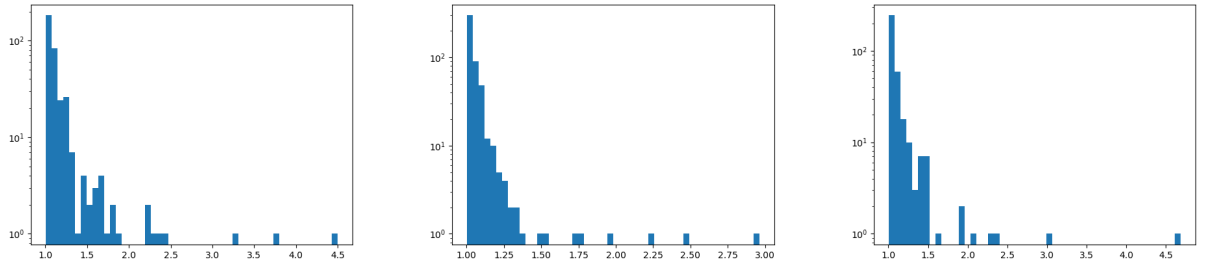
(a) *Cora* (b) *Citeseer* (c) *PubMed*

(d) *Texas* (e) *Wisconsin* (f) *Cornell*

Figure 7: Number of clusters inferred by DMPGM over the course of the MCMC chain.



(a) *Cora* (b) *Citeseer* (c) *PubMed*



(d) *Texas* (e) *Wisconsin* (f) *Cornell*

Figure 8: Histograms of the expected multiplicity of virtual edges formed in the EEGNN framework using each dataset.

	Cora	*Citeseer*	*PubMed*	*Texas*	*Wisconsin*	*Cornell*
dim_hidden	64	256	256	64	64	64
alpha	0.1	0.1	0.1	0.1	0.1	0.1
weight_decay	0.0005	0.0005	0.0005	0.0001	0.0005	0.0005
lr	0.01	0.01	0.01	0.01	0.01	0.01
dropout	0.6	0.7	0.6	0.5	0.5	0.5

(a) Hyperparameters for SGC and EE-SGC.

	Cora	*Citeseer*	*PubMed*	*Texas*	*Wisconsin*	*Cornell*
dim_hidden	64	64	64	64	64	64
alpha	0.1	0.1	0.1	0.1	0.1	0.1
lr	0.01	0.01	0.01	0.01	0.01	0.01
dropout	0.	0.	0.	0.	0.	0.
weight_decay1	0.005	0.005	0.005	0.005	0.005	0.005
weight_decay2	0.	0.	0.	0.	0.	0.
embedding_dropout	0.5	0.5	0.5	0.5	0.5	0.5

(b) Hyperparameters for APPNP and EE-APPNP.

	Cora	*Citeseer*	*PubMed*	*Texas*	*Wisconsin*	*Cornell*
dim_hidden	64	256	256	64	64	64
alpha	0.1	0.1	0.1	0.5	0.5	0.5
lamda	0.5	0.6	0.4	1.5	1.0	1.0
weight_decay1	0.01	0.01	0.0005	0.0001	0.0005	0.0001
weight_decay2	0.0005	0.0005	0.0005	0.0001	0.0005	0.0001
lr	0.01	0.01	0.01	0.01	0.01	0.01
dropout	0.6	0.7	0.6	0.5	0.5	0.5

(c) Hyperparameters for GCNII and EE-GCNII.

Table 3: Hyperparameters in the training.