Gaussian Regularization in Neural Graph Learning

Amzine Toushik Wasi[1], Taki Hasan Rafi[2], and Dong-Kyu Chae[2](✉)

[1] Shahjalal University of Science and Technology, Bangladesh
[2] Department of Computer Science, Hanyang University, South Korea
`azmine32@student.sust.edu`
`{takihr,dongkyu}@hanyang.ac.kr`

Abstract. Gaussian processes (GP), known for simplicity and flexibility, and the recent emergence of graph neural networks (GNNs) present promising avenues for semi-supervised learning on graph-structured data and beyond. Despite notable advancements in GNNs with a focus on neighborhood information, there are gaps in effectively integrating probability distributions and inter-entity relationships, as GNNs focus heavily on neighbors. Integrating probability distributions is essential to tackle noise and uncertainties. To address this issue, we present *Gaussian Regularization* in neural graph learning, which effectively incorporates Gaussian information to capture latent probabilistic distribution attributes from node embeddings based on the Gaussian Process, thereby boosting predictive capabilities. The process involves using a graph encoder to preprocess data, which is then encoded and used for effective aggregation and transformation of neighboring nodes. Gaussian Regularization works alongside any existing graph encoder model by encoding node representations into a Gaussian space to capture unique features. Logits are generated from this space, along with another set from an MLP, and then merged for final predictions. Extensive experiments using real-world benchmark datasets show that our approach outperforms several state-of-the-art GNN models and has a significant positive impact on off-the-shelf graph encoders and kernels, demonstrating its effectiveness and flexibility.

Keywords: Graph Neural Networks · Gaussian Process · Graph Representation Learning · Node Classification · Model Regularization

1 Introduction

Graph Neural Networks (GNNs) are a category of neural networks specialized in processing graph data structures, exhibiting notable effectiveness across diverse domains, including but not limited to node classification, link prediction, and graph classification tasks [13,14,28,27,24,1]. GNNs are explicitly devised to acquire comprehensive representations of nodes and edges within a given graph by assimilating pertinent information from their local neighbors [5,4,36,6]. For example, Graph Convolutional Networks (GCN) [12] applies convolution to aggregate node features, GraphSAGE [8] utilizes sampling and aggregation techniques to collect features from the local neighborhood of a node, and Graph Attention Networks (GAT) [3] utilizes self-attention to selectively aggregate information.

Although GNNs have achieved remarkable progress, they still possess certain limitations. One drawback of these methodologies is the omission of vital probabilistic distributions inherent in graphs, while only focusing on neighbour node data. Different graph networks heavily rely on such different relations containing specific probabilistic distribution characteristics, where probabilistic relations are as important as neighborhood information and it also reduces noises inherent in data [35,33]. Neglecting crucial information like this has detrimental effects, leading to diminished performance and compromised credibility of the model. One approach to address and harness probabilistic characteristics is to use the Gaussian Process (GP). It is a robust learning tool primarily used for regression and probabilistic classification tasks. Its working principle is based on the use of kernel functions, such as the Gaussian kernel, which considers the distance between data points to create a probabilistic model. GP possesses the unique advantage of providing probabilistic predictions which allows for the computation of empirical confidence intervals. This can assist in determining if a model requires refitting in areas of interest. Versatility is another key feature, with the flexibility to utilize various custom or standard kernels. They are capable of predicting highly calibrated probabilities unlike other models [10,26,7,30,22].

Researchers are adopting various approaches to concentrate on probabilistic distributions and relations inherent in graphs using Gaussian Processes. Ng et al. [19] presented a data-efficient Gaussian process-based Bayesian approach GGP, which works directly on graph data with a semi-supervised approach, adapted from off-the-shelf scalable variational inference algorithms for Gaussian processes. However, these methodologies do not actively utilize node aggregation methods and graph encoders. Additionally, a direct Gaussian approach to processing graph data imposes constraints on the models, restricting the utilization of highly effective graph convolutional encoders, which have proven pivotal in tackling a diverse array of graph problems. Furthermore, the existing GNN models with different capabilities usually lack seamless compatibility with already running GNN pipelines in production, as they fail to complement the established architectures. Should one opt to implement these models, a replacement of the current running system becomes imperative in most cases.

While GP in GNNs have been extensively explored in recent research, there are still gaps to be addressed. Existing approaches directly incorporate or infuse GP within GNN models during message passing or information sharing, effectively reducing uncertainties. However, this direct integration restricts GP's compatibility with other GNN architectures such as SAGE, GAT, or SGC, necessitating the development of new architectures (like GP-SAGE or GP-GAT) specifically tailored for these combinations. The key distinction with our model is that it allows seamless integration of GP with various graph encoders, unlike existing approaches that develop dedicated GP-based GNN encoders. This limitation of existing works also hampers the practical applicability of existing methodologies within established GNN pipelines, necessitating system replacements.

To overcome the limitations discussed earlier, we present a simple yet effective approach, Gaussian Regularization (GR), in Neural Graph Learning. GR leverages a Gaussian Process (GP) (using Nyström approximation [29] to reduce computational complexity) to utilize the probabilistic properties of data, enhancing prediction accuracy. By employing GP, our model reduces noise and focuses on critical node information, achieving robust predictions and outperforming other models with similar training requirements. Instead of directly feeding raw graphs into GP, our pipeline first encodes the data through a graph encoder, offering flexibility to use various encoders optimized for aggregating and transforming node information. GR employs a single-layer MLP to generate initial logits, which are then combined with GP output logits via a weighted sum, stabilizing the model and refining predictions. Experimental results show that GR models outperform state-of-the-art graph-based predictors.

Our contributions are summarized into four folds:

- We introduce Gaussian Regularization for GNNs, a simple yet powerful module designed to use Gaussian information from node embeddings in a very effective manner. Integrating into any GNN pipeline, our model can drastically reduce training iterations and improve the prediction capabilities of any existing graph encoder at the same time.
- We leverage Gaussian Process to exploit latent probabilistic distribution characteristics and relations very efficiently and remove uncertainties and noises effectively. We utilize this information to enhance prediction scores by regularizing logits using a computationally efficient approach.
- Our study extensively investigates the impact of different graph encoders, Gaussian kernels, depths, and data types on Gaussian Regularization, while analyzing the reasons behind observed discrepancies through extensive analysis.
- Our method is rigorously evaluated on diverse benchmark graph databases and baselines. Experimental results show that Gaussian Regularization is highly capable of utilizing Gaussian probabilistic characteristics and improving predictions in a very effective manner.

2 Related Works

A Gaussian process (GP) is a type of stochastic process utilized for modeling random variables that are indexed by either time or space. But, general GP method has some issue in scalability and complexity. To solve the limitations of sparse Gaussian Process (GP) classification with millions of data, scalable variational GP [9] has been developed on the basis of a variational inducing point framework along with Nyström approximation [29] used in our model, where it can be used to facilitate classification in complex issues. The strategy implemented by Niu et al. [20] involves integrating inductive bias of GNNs into GPs with the objective of enhancing graphical data's predictive performance, where

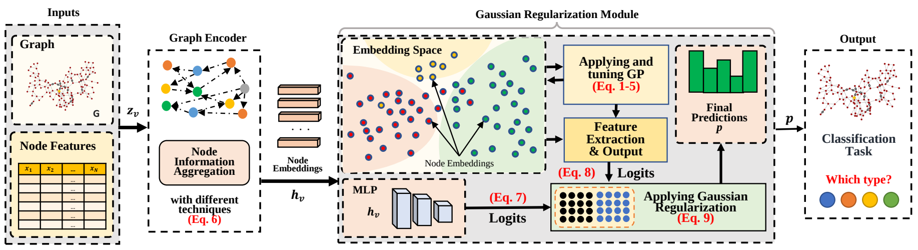

Fig. 1. Our proposed framework for Gaussian Regularization on GNNs.

the GNN's proficiency is comparable to some GP when its layers are indefinitely wide. They also propose an automated method for creating covariance kernels that are motivated by similarity and create example kernels that correspond to a number of GNNs, achieving superior classification and regression performance as well as improved computation. Disordered Graph Convolutional Neural Network (DGCNN) by Wu et al. [31] involves a preprocessing layer, Disordered Graph Convolution Layer (DGCL) where the DGCNN expands the CNN. The convolution kernel and the nodes surrounding the graph are mapped employing a mixed Gaussian function by the DGCL. The CNN receives its data from the DGCL's output and DGCNN minimizes information loss during graph transformation by accepting randomly scaled and unstructured neighborhood graph frameworks as the receptive fields of CNNs. Opolka et al. [21] presents GCLGP, a variational inducing point technique that positions pseudo inputs within a graph-structured domain to scale the Gaussian process model to large graphs.

While recent work has integrated GPs into GNNs to manage uncertainties, most approaches embed GP directly in the GNN's message-passing process. This integration limits compatibility with a range of GNN architectures, requiring specific models stated above, which reduces flexibility. Our approach stands out because it allows GP to work with any graph encoder, eliminating the need for specialized GP-based models. This flexibility makes our model adaptable across various GNN pipelines, enhancing scalability and practical use without the need for system overhauls, thus boosting predictive robustness and efficiency.

3 Gaussian Regularization for Neural Graph Learning

Let a graph denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ represents the set of nodes $\{v_1, v_2, v_3, \cdots v_N\}$, $\mathcal{E}$ denotes the set of edges $\{e_1, e_2, e_3, \cdots e_M\}$, where N, M is the number of nodes and edges, respectively. $\mathcal{A}$ is an adjacency matrix of size $N \times N$ representing graph $\mathcal{G}$, where $\mathcal{A}_{ij} = 1$ if there is an edge between v_i and v_j. Each node v has its own feature $x_i \in \mathbb{R}^F$, and the node feature matrix of the whole graph is denoted as $\mathcal{X} \in \mathbb{R}^{N \times F}$, F being feature vector length. A mapping function $g : v \to z_v$ maps a node in a graph to a representation vector, where the vector $z_v \in \mathbb{R}^d$ represents the position of the node in a continuous latent space of dimensions d. The full set of representation vectors is defined by Z containing

$\{z_1, z_2, z_3, \cdots z_N\}$. In our task, the graph encoder generates node embeddings h_v from z_v. These embeddings are passed on to our Gaussian Regularization module to obtain predictions p (see Figure 1).

3.1 Gaussian Process

A Gaussian Process (GP), denoted as $f(\cdot)$, is a mathematical framework used to model random variables indexed by time or space, characterized as an infinite collection of random variables where any finite subset follows a joint Gaussian distribution. A GP is fully defined by its mean function $m(\cdot)$, which captures the overall trend of the data, and its covariance kernel function $K(\cdot)$, which determines the dependencies between different inputs x and y that index the process. The output of the kernel function, represented as K, quantifies the degree of similarity or correlation between these inputs. Additionally, the set of hyperparameters θ governs the likelihood function of the GP, influencing the behavior of the covariance structure. Using the mean function and covariance kernel, the GP can be succinctly expressed as $f(\cdot) \sim \mathcal{GP}(m(\cdot), K(\cdot))$, encapsulating both the trend and the interdependencies within the data. By employing this mean and covariance kernel, we can succinctly denote the GP as follows:

$$f(x) \sim \mathcal{GP}\left(m(\cdot), K\right) \tag{1}$$

GPs are popular priors in Bayesian machine learning due to their versatile support, tractable posterior (under certain conditions), and other desirable properties. By combining GPs with an appropriate likelihood function, such as in regression or classification, one can create models that effectively handle uncertainties and overfitting through Bayesian smoothing. However, when dealing with non-Gaussian likelihoods (e.g., in classification tasks), obtaining the posterior process becomes analytically challenging and requires approximations. Nonetheless, GPs remain connected to the observed data through the likelihood function as follows:

$$y_n \mid f\left(X\right) \sim \log p(y_n \mid X, \theta) \tag{2}$$

Gaussian Process Model For a general Gaussian Process model, input $X = (x_1, \ldots, x_N)^\top$ and output $y = (y_1, \ldots, y_N)$ defined with $p(y \mid f) = \mathcal{N}\left(y \mid f, \sigma^2 I\right)$ and $p(f \mid X) = \mathcal{N}(f \mid 0, K(X, X))$, prediction on a test point given the observed data and the model parameters:

$$p\left(f_* \mid X_*, y, X, \theta\right) = \quad \mathcal{N}\left(f_* \mid K_*\left(K + \sigma^2 I\right)^{-1} y, K - K_*\left(K + \sigma^2 I\right)^{-1} K_*^\top\right) \tag{3}$$

where K is the covariance matrix, generated by kernels (kernel outputs). $*$ denotes function values at test points. σ is the standard deviation, l is the characteristic length-scale [30]. In this work, Matern kernel is used for adding regularization in GP.

$$K = k_M\left(x_i, x_j\right) = \frac{1}{\Gamma(\nu)2^{\nu-1}}\left(\frac{\sqrt{2\nu}}{l} d\left(x_i, x_j\right)\right)^\nu K_\nu\left(\frac{\sqrt{2\nu}}{l} d\left(x_i, x_j\right)\right) \tag{4}$$

Here $d(\cdot,\cdot)$ is the Euclidean distance, ν controls smoothness, $K_\nu(\cdot)$ is a modified Bessel function and $\Gamma(\cdot)$ is the gamma function.

To get log-likelihood estimate which helps it remain connected to the observed data, the log-pdf of multi-variate normal distribution is used as follows:

$$\log p(y \mid X, \theta) = -\frac{1}{2}\log\left|2\pi\left(K+\sigma^2 I\right)\right| - \frac{1}{2}y^\top\left(K+\sigma^2 I\right)^{-1}y \tag{5}$$

Eq. 1 summarises all these Gaussian Process works into one callable equation. To gain a comprehensive understanding of Gaussian Processes and kernel functions, we refer to [30].

3.2 Graph Encoders

A Graph Encoder is a specialized neural network designed to process graph-structured data, drawing inspiration from Convolutional Neural Networks (CNNs) and graph embedding techniques. Its primary function to handle graph data structures and facilitate various tasks related to nodes, edges, and graphs themselves. It takes initial input consists of a graph accompanied by node features. Then, node features are combined with the features of neighboring nodes through aggregation operations. The aggregated features undergo a non-linear transformation, resulting in updated node features. The final output of the Graph Encoder comprises a new set of node features called node embeddings, which can be utilized for downstream tasks like node classification, link prediction, or graph classification. This iterative process can be repeated multiple times, with each layer's output serving as the subsequent layer's input. This enables the model to capture information from distant parts of the graph, enhancing its ability to understand complex relationships within the data.

In this paper, our model is designed to handle any type of graph encoder. Our graph encoder uses a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, takes input z_v, aggregates information from the nodes using different aggregation functions (f_A), transforms (f_T) it using some model parameters, and outputs a node embedding h_v. A general formula of graph encoders can be noted as:

$$h_v = f_T(f_A(z_v)) \tag{6}$$

Our approach offers a significant advantage allowing for the utilization of any graph encoder to process primary information. This compatibility enables seamless integration with existing graph encoders, including those already in production, while simultaneously enhancing their prediction capabilities.

3.3 Gaussian Regularization Module

The Gaussian Regularization (GR) module handles the overall Gaussian Process. It takes h_v as input from the previous layer of graph encoder, and applies a Gaussian probabilistic approach to it. It tries to gather and group node embeddings together using different calculations described in Section 3. In the GR module, we first create a set of logits using MLP, as per the regular way. Then, we generate another set of logits from the Gaussian process and regularize the first set using the second set using a weighted sum function.

MLP Logits Generation. A linear layer takes h_v and outputs logits p_{MLP} as the initial prediction.

$$p_{MLP} = MLP(h_v) \tag{7}$$

Gaussian Process Logits Generation. Gaussian Process described in Section 3.1 takes H (train node features) as input, fits the model using y (true labels), and outputs logits p_{GP}, using Eq. 1 which summarises the whole Gaussian Process model used in our work. Here, $H = (h_1, h_2, \ldots, h_N)^\top$, where h_v is the graph-encoded node embedding vector (from graph encoder mentioned in Eq. 6) of any node v from total N nodes. H corresponds to input X in Section 3.1.

$$p_{GP} = GP(H) \tag{8}$$

Here, h_v is the input x of the GP function $f(x)$ in Eq. 1.

Applying Regularization. p_{MLP} (Eq. 7) and p_{GP} (Eq. 8) are combined using a weighted some approach, where α is a hyperparameter controlling the weight of p_{GP}. The final logits output p can be defined as:

$$p = softmax(p_{MLP} + \alpha p_{GP}) \tag{9}$$

Argmax of p is the final prediction label.

3.4 Training Objective

In our training process, we utilize cross entropy (CE) loss $\mathcal{L}_{ce}$. As the experiments in our work is multi-class classifications, multi-class CE loss is applied here, which designed to minimize the difference between the predicted probabilities of different classes of nodes.

$$\mathcal{L}_{ce} = \frac{\sum_{n=1}^{N} l_n}{N}, \quad l_n = -\sum_{c=1}^{C} w_c \log \frac{\exp(p_{n,c})}{\sum_{i=1}^{C} \exp(p_{n,i})} y_{n,c} \tag{10}$$

where p_n represents the output of the n-th data point, where i ranges from 1 to n, and true label of the n-th data point is denoted by y_n. w is the weight (all weights are kept equal in our work), C is the number of classes and and N spans the minibatch dimension. For different tasks and models, different loss functions can be applied, too.

4 Experimental Settings

Our module is capable of handling different tasks on diverse graph datasets. In this section, we evaluate our proposed model on classification tasks. To verify the effectiveness of our **Gaussian Regularization (GR)** module, our aim is to answer the following research questions (RQs):

Procedure: Gaussian Regularization in Neural Graph Learning Process

```
Input: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, node features $\mathcal{X} \in \mathbb{R}^{N \times F}$
Output: predictions $p$
// Data Processing
1: Create $z_v \in \mathbb{R}^d$ by a mapping function $g : v \to z_v$
// Model Training
2: Initialize all parameters for the model;
3: while $\mathcal{L}_{ce}$ does not converge do
4:     Get node embeddings from the encoder using
       $h_v = f_T(f_A(z_v))$ (Eq. 6);
5:     Get primary logits by MLP on $h_v$ using
       $p_{MLP} = MLP(h_v)$ (Eq. 7);
6:     Train the Guassian Process GP on $h_v$ (Eq. 1-5,11);
7:     Extract features and generate logits using
       Guassian Process $p_{GP} = GP(h_v)$ (Eq. 8);
8:     Apply Gaussian Regularization using
       $p = softmax(p_{MLP} + \alpha p_{GP})$ (Eq. 9);
9:     Calculate the loss function $\mathcal{L}_{ce}$ (Eq. 10);
10:    Update all parameters by stochastic gradient descent;
11: end while
```

- **RQ 01:** How is the performance of this model? Does it outperforms existing GNNs?
- **RQ 02:** How is the performance of this model if the GR module is removed?
- **RQ 03:** If and how does each Guassian Process kernel improves the performance?
- **RQ 04:** How do different graph encoders respond to GR?
- **RQ 05:** How scalable is the GR module?
- **RQ 06:** How is the model affected in terms of its stability and smoothness properties?

Datasets. The experiments encompass a different benchmark graph-based datasets for classification tasks. Cora, CiteSeer, PubMed [34] incorporate scientific papers organized within a citation network; Amazon Computers and Photo [25] are from the Amazon co-purchase graph and Reddit [8] is for predicting the community of online posts based on user comments. These datasets were divided into a 6:2:2 ratio for training, validation, and testing respectively. For more comprehensive information about the datasets, please refer to Table 1.

Graph Encoders. As our model is suitable for any graph encoder, we have evaluated our model's compatibility with various graph encoders developed by different researchers, each offering unique perspectives. GAT [3] employs an attention mechanism for node feature aggregation, while GATv2 [3] addresses the static attention problem of GATConv. GCN [12] utilizes semi-supervised learning for node feature aggregation, and SAGE [8] operates within an inductive framework. GraphConv [18] relies on the 1D Weisfeiler-Leman graph isomorphism, while GEN [15] leverages deeper GCN to train deep GCNs effectively.

Table 1. Dataset statistics.

Name of the Dataset	Nodes	Edges	Features	Classes
Cora [34]	2,708	5,429	1,433	7
CiteSeer [34]	3,327	4,732	1,345	6
PubMed [34]	19,717	88,648	500	3
Amazon Computers [25]	13,752	491,722	767	10
Amazon Photo [25]	7,650	238,162	745	8
Reddit [8]	232,965	114,615,892	602	41

Lastly, SGC [32] adopts a GCN-based model with a strategy to reduce complexity. These methods offer diverse approaches to graph-based tasks.

Implementation Details. For all our experiments, we used a fixed seed value of 42. We apply a learning rate of 0.01 along with a weight decay hyper-parameter of 5e-4. The output length of the graph encoder, which represents the node embeddings, is set to 256. In Table 3, all the models without GR are trained for 50 epochs, with early stopping if loss increases or the accuracy drops 4 times, and the models with GR are trained for 10 epochs. The models without GR converged in around 20-30 epochs, and the models with GR converged in 3-6 epochs. α in Eq. 9 is set to $0.6 \sim 0.7$, number of nodes in subset for Nyström approximation (M) is used $0.6 \sim 0.8N$.

5 Results and Analyses

5.1 RQ1: Performance comparison with SOTA models

The accuracy of different models in these datasets is presented in Table 2. Among the existing GP-based models, GGP and GCNGP demonstrated strong performance. Notably, GCNGP outperformed base GGP with an accuracy of 82.8% on Cora. Our Graph Regularization based model with GCN encoder exhibited exceptional accuracy rates. It achieved remarkable scores in all datasets, surpassing all other models. Some of the scores are missing due to official code availability (DCGN [2], GGP [19]), and reproducibility issues with different datasets' data-loaders (GCNGP [20], EEGNN [16], GAT [3], MoNet [17]).

5.2 RQ2: Effectiveness of GR Module

Based on the findings presented in Table 3, the removal of the GR module significantly impacts the performance of the model in classification tasks. In the Cora dataset, utilizing the GR module with a GAT encoder and RBF kernel resulted in an accuracy of 82.8%. However, when the GR module was removed, the accuracy dropped to 58.4%. Similarly, in the PubMed dataset, employing the GR module with a GCN encoder and Matern Kernel achieved an accuracy of 83.1%. Nevertheless, removing the GR module led to a decrease in accuracy, resulting in a score of 76.8%. Overall, most models exhibited negative reactions

Table 2. Accuracy (%) (↑) comparisons.

Method	Cora	CiteSeer	PubMed	Amazon Computer	Reddit	Amazon Photo
DW [23]	67.2	43.2	78.8	62.7	69.1	69.1
DCNN [2]	76.8	73.3	76.8	-	-	-
GCN [12]	81.5	70.3	76.8	82.6	93.3	91.2
GGP [19]	80.9	69.7	77.1	-	90.6	-
SGC [32]	81.0	68.9	75.8	77.2	92.6	91.1
GAT [3]	81.7	68.8	77.7	78.0	-	85.7
MoNet [17]	81.7	73.9	73.9	83.5	-	91.2
GCNGP [20]	82.8	70.9	79.6	-	94.6	-
EEGNN [16]	85.5	72.2	79.9	-	-	-
GR-SGC	**85.8**	**77.8**	**82.6**	**90.2**	**94.6**	**94.1**
GR-GCN	**86.2**	**78.1**	**83.1**	**90.9**	**95.1**	**94.8**

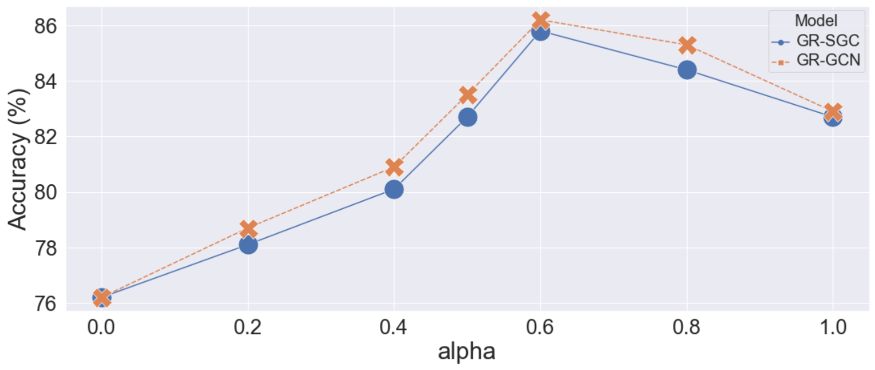

Fig. 2. Comparison of performance with different value of α (PubMed).

when the GR module was eliminated, indicating its importance in maintaining performance.

5.3 RQ3: Comparison of Different GR Setups

Impact of different GP kernels. Use of different kernels has its own effects, as shown in Table 3. Among the various kernels considered, our choice of the regularized Matern kernel yielded the best results. It outperformed the RQ kernel by 4-6% in terms of accuracy and exceeded the RBF kernel by 2-4% in accuracy.
Impact of Hyperparameter α. Figure 2 shows the performance of two best-performing GR-based models, GR-SGC and GR-GCN, at different values of the α hyperparameter using the Cora dataset. As α increases from 0 to 0.6, both models generally exhibit an improvement in their classification accuracy. At $\alpha =$ 0.6, both models achieve their highest accuracy. If alpha continues to increase beyond 0.6, the accuracy for both models starts to decline slightly. The

Table 3. Comparison of different experimental setups.

Setup		Without GR			GR (RBF Kernel)			GR (RQ kernel)			GR (Matern kernel)		
Dataset	Enoder	Acc.	Prec.	F1	Acc.	Prec.	F1	Acc.	Prec.	F1	Acc.	Prec.	F1
Cora	GAT	58.4%	59.1%	.688	82.8%	82.9%	.839	80.3%	80.4%	.814	84.1%	84.2%	.852
	GATv2	64.6%	74.0%	.735	83.5%	84.4%	.844	81.0%	81.9%	.819	84.8%	85.7%	.857
	GCN	76.2%	80.7%	.812	84.6%	85.1%	.851	82.1%	82.6%	.826	**86.2%**	86.4%	.864
	SAGE	71.2%	77.4%	.765	84.1%	84.7%	.847	81.6%	82.2%	.822	84.9%	86.0%	.860
	GraphConv	70.0%	75.6%	.768	84.0%	84.6%	.847	81.5%	82.1%	.822	85.1%	85.9%	.860
	GEN	46.6%	53.5%	.591	81.7%	82.4%	.829	79.2%	79.9%	.804	83.0%	83.7%	.842
	SGC	76.2%	80.5%	.811	84.6%	85.1%	.851	82.1%	82.6%	.826	**85.8%**	86.4%	.864
CiteSeer	GAT	53.8%	71.6%	.723	76.4%	78.2%	.782	74.6%	76.4%	.764	75.9%	77.7%	.797
	GATv2	60.6%	80.2%	.778	77.1%	79.0%	.788	75.3%	77.2%	.770	76.6%	78.5%	.813
	GCN	69.0%	82.2%	.822	77.9%	79.2%	.792	76.1%	77.4%	.774	**78.1%**	78.7%	.817
	SAGE	67.6%	83.6%	.830	77.8%	79.4%	.793	76.0%	77.6%	.775	77.3%	78.9%	.818
	GraphConv	63.4%	76.7%	.791	77.3%	78.7%	.789	75.5%	76.9%	.771	76.8%	78.2%	.804
	GEN	34.2%	58.5%	.607	74.4%	76.9%	.771	72.6%	75.1%	.753	73.9%	76.4%	.796
	SGC	69.0%	83.3%	.828	77.9%	79.3%	.793	76.1%	77.5%	.775	**77.8%**	78.8%	.798
PubMed	GAT	63.4%	73.2%	.736	80.3%	80.3%	.804	78.8%	79.8%	.799	80.3%	81.3%	.814
	GATv2	62.6%	71.2%	.734	79.3%	80.1%	.803	78.8%	79.6%	.798	80.3%	81.1%	.813
	GCN	76.8%	77.7%	.746	80.7%	80.8%	.805	80.2%	80.3%	.800	**83.1%**	81.8%	.815
	SAGE	77.0%	76.8%	.752	80.7%	80.7%	.805	80.2%	80.2%	.800	81.7%	81.7%	.815
	GraphConv	75.6%	75.3%	.744	80.2%	80.5%	.804	80.1%	80.0%	.799	81.6%	81.5%	.814
	GEN	68.2%	70.4%	.669	79.8%	80.0%	.797	79.3%	79.5%	.792	80.8%	81.0%	.807
	SGC	76.0%	76.5%	.738	80.6%	80.7%	.804	80.1%	80.2%	.799	**82.6%**	81.7%	.814
Compu.	GAT	59.5%	79.1%	.777	83.9%	85.7%	.862	77.1%	78.9%	.796	86.3%	87.9%	.882
	GATv2	68.8%	87.9%	.840	84.8%	86.9%	.886	79.2%	80.9%	.791	87.3%	89.1%	.891
	GCN	74.5%	87.9%	.879	87.1%	86.8%	.873	79.0%	81.2%	.801	**90.9%**	90.3%	.911
	SAGE	73.6%	91.5%	.907	87.9%	89.6%	.882	78.7%	80.6%	.813	87.8%	89.7%	.890
	GraphConv	70.6%	83.4%	.847	86.7%	86.0%	.889	79.3%	80.4%	.803	87.2%	88.7%	.888
	GEN	39.6%	66.2%	.682	82.8%	85.5%	.850	76.7%	77.8%	.7925	84.1%	86.7%	.873
	SGC	74.5%	90.4%	.904	85.2%	87.1%	.891	79.5%	81.6%	.797	**90.2%**	89.0%	.924
Photo	GAT	69.1%	72.5%	.726	86.7%	87.4%	.877	84.9%	86.2%	.858	88.4%	90.0%	.901
	GATv2	74.9%	75.6%	.858	86.3%	88.4%	.892	84.7%	87.3%	.876	92.2%	89.7%	.917
	GCN	83.8%	84.7%	.812	87.2%	89.3%	.885	85.4%	87.7%	.879	**94.8%**	94.5%	.959
	SAGE	83.6%	81.0%	.811	88.3%	89.4%	.886	86.0%	87.7%	.874	89.4%	91.0%	.902
	GraphConv	76.9%	75.0%	.770	86.8%	89.1%	.887	84.4%	86.7%	.864	89.5%	90.7%	.914
	GEN	47.6%	46.4%	.490	84.4%	87.6%	.873	82.7%	85.5%	.853	85.5%	88.1%	.883
	SGC	83.4%	82.3%	.847	86.9%	89.5%	.896	86.0%	88.1%	.874	**94.1%**	93.9%	.953
Reddit	GAT	70.3%	75.8%	.756	88.0%	89.6%	.895	86.7%	88.5%	.881	90.6%	91.2%	.922
	GATv2	76.2%	79.8%	.775	88.6%	90.3%	.903	86.7%	89.1%	.891	91.1%	91.9%	.933
	GCN	85.5%	86.0%	.854	89.7%	90.5%	.905	87.8%	89.0%	.892	**95.1%**	95.2%	.957
	SAGE	84.7%	83.4%	.839	89.9%	91.0%	.908	87.4%	89.0%	.887	91.8%	92.8%	.923
	GraphConv	79.0%	81.2%	.810	88.8%	90.3%	.903	86.8%	88.5%	.886	91.7%	92.3%	.925
	GEN	49.8%	51.5%	.609	86.2%	89.2%	.884	84.7%	86.8%	.877	88.0%	89.8%	.907
	SGC	84.7%	85.5%	.853	89.1%	90.5%	.914	88.2%	89.5%	.895	**94.6%**	94.6%	.948

Acc. denotes Accuracy (↑), Prec. denotes Precision (↑) and F1 denotes F1 score (↑).

optimal value of α is 0.6. For new encoders or datasets, it is needed to be tuned appropriately for optimal performance.

5.4 RQ4: Impact of Graph Encoders

Table 3 showcases the performance of various encoders in different experimental setups. The results indicate that both GCN and SGC-based encoders consistently performed well across all datasets, regardless of the presence of the GR module. For instance, in the Cora dataset without the GR module, GCN achieved the highest accuracy of 76.2%. With the inclusion of the GR module, GCN continued to excel with scores of 84.6% using RBF kernel, 82.1% using RQ kernel, and a maximum of 86.2% using Matern kernel. Following closely behind, SGC Graph encoder secured the second position. Similar trends were observed in the CiteSeer and PudMed datasets too, with GCN outperforming other encoders, and SGC encoder ranking second. GAT performs poorly with its attention-focused mechanism in all setups.

5.5 RQ5: Scalability Analysis

We note that all our experiments were performed on a GPU server equipped with NVIDIA RTX 3090 GPU with 128 GB RAM, along with an Intel Core i9-12900K CPU.

Impact of GR with regard to numbers of classes, nodes, and edges. In our analysis of scalability, we examine the running time of our models across different numbers of classes, nodes, and edges (see Figure 3). The running times are normalized between 0 and 1. We observe that the running time of the models using our Gaussian Regularization (GR) increased as the number of classes grew. This can be attributed to the one-against-all method employed by the Gaussian Process in multiclass classification, which in turn increases the runtime of the model. When considering the scalability in terms of the number of nodes or edges, the impact on GR was nearly linear considering the phenomenon caused by the number of classes. For classification with more than six classes, parallel computing or other scalability techniques can be applied.

Impact of GR on Running Time. The runtime results for the GCN and SGC graph encoders with and without the GR module on the Cora, CiteSeer, and PubMed datasets are presented in Figure 5. From Figure 5(a), it is apparent that the GCN without the GR module exhibits the fastest running time across all models and datasets. Interestingly, in the PubMed dataset, the runtime for all models to complete one epoch is very similar. The chart shows that incorporating the GR module into the pipeline usually increases the overall running time of an epoch. It is also apparent from Figure 5(b) that the GR module converges and achieves maximum performance faster, within a relatively short iteration of approximately 3-5 epochs, whereas other modules require significantly more epochs to reach comparable levels of accuracy; for example, SGC requires a lot more iterations and time to converge. Though GR models can perform 1 epoch very

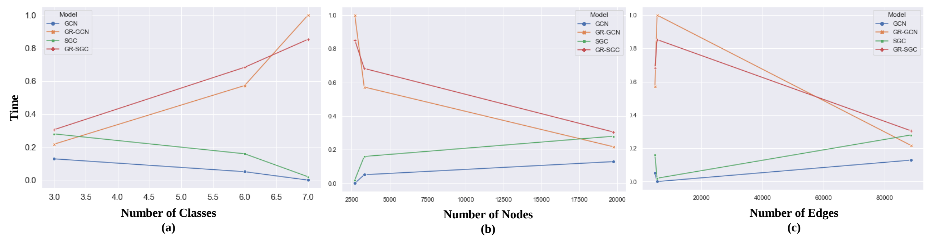

Fig. 3. Scaling of the running time with respect to graph statistics (number of classes, nodes, and edges).

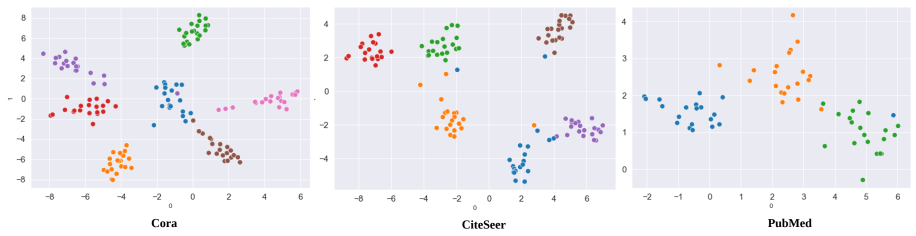

Fig. 4. Visualization of the feature representations of Cora, CiteSeer, and PubMed obtained from GR-GCN in a 2-D space.

fast, they require a lot of epochs, ultimately increasing training time. Considering everything, we can conclude that our model can run with lower computation time in classification tasks with fewer classes. In these situations, it converges faster as it needs to perform less one-against-all individual classification tasks. We excluded datasets with more than 7 classes from this scalability analysis, as the runtime is too much to visualize and analyze.

5.6 RQ6: Smoothness and Stability

Smoothness and stability characteristics of our model is imposed by the Matern kernel (Eq. 4) used in our model. Compared to the original RBF kernel and Rational Quadratic (RQ) kernel which don't have smoothness and stability regularization [11], our design has some advantages in this regard. In Table 3, there is a comparison of evaluation scores between RBF kernel and Matern kernel. The scores show that the Matern kernel achieves around 2-5% more accuracy, precision, and F1 score than the RBF and RQ kernel; showing the effectiveness of the smoothness and stability regularizer used in our design.

6 Discussion

One limitation of the model lies in its efficacy when applied to classification tasks involving a high number of classes. This challenge is thoroughly addressed in Research Question 5, as discussed in Section 5.5. Our model employs a strategy

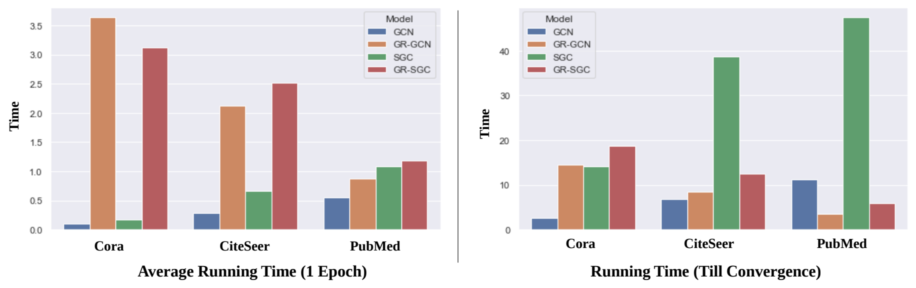

Fig. 5. Running time comparison.

that relies on multiple one-vs-all classification approaches and combines their parameters to achieve multi-class classification. However, when confronted with a substantial number of classes, scalability becomes an issue. Nevertheless, it is noteworthy that the model exhibits remarkable effectiveness in binary classification or tasks involving up to three classes, surpassing the performance of most other active GNN models in such scenarios. Figure 4 demonstrates the presence of distinct groups or classes in the 2D space of three datasets: Cora, CiteSeer, and PubMed analyzing the node embeddings. In the Cora and CiteSeer datasets, we observe 7 and 6 distinct groups with a few exceptions.

Future endeavors following this work may involve the exploration of potential solutions to mitigate this challenge with higher number of classes in multi-class classification, potentially through the development and application of more stringent regularization techniques and robust kernel methodologies.

7 Conclusion

In this paper, we present the integration of Gaussian Regularization into graph neural learning. Our approach leverages Gaussian space to capture and propagate latent probabilistic distribution characteristics from node embeddings. The proposed architecture is compatible with any existing graph encoder, facilitating data aggregation by gathering essential neighbor information for effective functional modeling and transformation of surrounding nodes using Gaussian processes (GP). Extensive experimentation on several benchmark datasets, graph encoders, and baseline models showcase that our module is capable of achieving remarkable performance, indicating its flexibility, feasibility, and overall generalization.

Acknowledgments. This work was partly supported by (1) the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (RS-2024-00345398) and (2) the Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government (MSIT)(RS-2020-II201373,Artificial Intelligence Graduate School Program (Hanyang University)).

References

1. Agarwal, A., et al.: Fs-dag: Few shot domain adapting graph networks for visually rich document understanding. In: COLING: Industry Track (2025)
2. Atwood, J., Towsley, D.: Diffusion-convolutional neural networks. Advances in neural information processing systems **29** (2016)
3. Brody, S., Alon, U., Yahav, E.: How attentive are graph attention networks? In: International Conference on Learning Representations (2022)
4. Bronstein, M.M., Bruna, J., Cohen, T., Veličković, P.: Geometric deep learning: Grids, groups, graphs, geodesics, and gauges (2021)
5. Bronstein, M.M., Bruna, J., LeCun, Y., Szlam, A., Vandergheynst, P.: Geometric deep learning: Going beyond euclidean data. IEEE Signal Processing Magazine **34**(4), 18–42 (2017)
6. Chen, M., Wei, Z., Huang, Z., Ding, B., Li, Y.: Simple and deep graph convolutional networks. In: International conference on machine learning. pp. 1725–1735 (2020)
7. Frazier, P.I.: A tutorial on bayesian optimization (2018)
8. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. vol. 30 (2017)
9. Hensman, J., Matthews, A., Ghahramani, Z.: Scalable variational gaussian process classification. In: Artificial Intelligence and Statistics. pp. 351–360. PMLR (2015)
10. Kac, M., Siegert, A.J.F.: An Explicit Representation of a Stationary Gaussian Process. The Annals of Mathematical Statistics **18**(3), 438 – 442 (1947)
11. Kanagawa, M., Hennig, P., Sejdinovic, D., Sriperumbudur, B.K.: Gaussian processes and kernel methods: A review on connections and equivalences (2018)
12. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: International Conference on Learning Representations (2017), `https://openreview.net/forum?id=SJU4ayYgl`
13. Lee, J., Chae, D.K.: Multi-view mixed attention for contrastive learning on hypergraphs. In: Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 2543–2547 (2024)
14. Lee, J., Park, B., Chae, D.K.: Duogat: Dual time-oriented graph attention networks for accurate, efficient and explainable anomaly detection on time-series. In: Proceedings of the 32nd ACM International Conference on Information and Knowledge Management. pp. 1188–1197 (2023)
15. Li, G., Xiong, C., Thabet, A., Ghanem, B.: Deepergcn: All you need to train deeper gcns (2020)
16. Liu, Y., Qiao, X., Wang, L., Lam, J.: Eegnn: Edge enhanced graph neural network with a bayesian nonparametric graph model. In: International Conference on Artificial Intelligence and Statistics. pp. 2132–2146. PMLR (2023)
17. Monti, F., Boscaini, D., Masci, J., Rodola, E., Svoboda, J., Bronstein, M.M.: 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 (2017)
18. Morris, C., Ritzert, M., Fey, M., Hamilton, W.L., Lenssen, J.E., Rattan, G., Grohe, M.: Weisfeiler and leman go neural: Higher-order graph neural networks (2021)

19. Ng, Y.C., Colombo, N., Silva, R.: Bayesian semi-supervised learning with graph gaussian processes. Advances in Neural Information Processing Systems **31** (2018)
20. Niu, Z., Anitescu, M., Chen, J.: Graph neural network-inspired kernels for gaussian processes in semi-supervised learning. In: The Eleventh International Conference on Learning Representations (2023)
21. Opolka, F.L., Liò, P.: Graph convolutional gaussian processes for link prediction (2020)
22. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
23. Perozzi, B., Al-Rfou, R., Skiena, S.: 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)
24. Seong, E., Lee, H., Chae, D.K.: Self-supervised framework based on subject-wise clustering for human subject time series data. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 38, pp. 22341–22349 (2024)
25. Shchur, O., Mumme, M., Bojchevski, A., Günnemann, S.: Pitfalls of graph neural network evaluation (2019)
26. Snoek, J., Larochelle, H., Adams, R.P.: Practical bayesian optimization of machine learning algorithms. Advances in neural information processing systems **25** (2012)
27. Wasi, A.T., Rafi, T.H., Islam, R., Chae, D.K.: BanglaAutoKG: Automatic Bangla knowledge graph construction with semantic neural graph filtering. In: LREC-COLING 2024. pp. 2100–2106 (May 2024)
28. Wasi, A.T., Rafi, T.H., Islam, R., Karlo, S., Chae, D.K.: Cadgl: Context-aware deep graph learning for predicting drug-drug interactions (2024)
29. Williams, C., Seeger, M.: Using the nyström method to speed up kernel machines. Advances in neural information processing systems **13** (2000)
30. Williams, C.K., Rasmussen, C.E.: Gaussian processes for machine learning, vol. 2. MIT press Cambridge, MA (2006)
31. Wu, B., Liu, Y., Lang, B., Huang, L.: Dgcnn: Disordered graph convolutional neural network based on the gaussian mixture model. Neurocomputing **321**, 346–356 (2018)
32. Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., Weinberger, K.: Simplifying graph convolutional networks. In: International conference on machine learning. pp. 6861–6871 (2019)
33. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? In: International Conference on Learning Representations (2019)
34. Yang, Z., Cohen, W., Salakhudinov, R.: Revisiting semi-supervised learning with graph embeddings. In: International conference on machine learning. pp. 40–48 (2016)
35. Yoon, K., Liao, R., Xiong, Y., Zhang, L., Fetaya, E., Urtasun, R., Zemel, R., Pitkow, X.: Inference in probabilistic graphical models by graph neural networks. In: ICLR 2018 Workshop (2019)
36. Zhang, M., Chen, Y.: Link prediction based on graph neural networks. Advances in neural information processing systems **31** (2018)