DropMessage: Unifying Random Dropping for Graph Neural Networks

Taoran Fang[1], Zhiqing Xiao[1], Chunping Wang[2], Jiarong Xu[3], Xuan Yang[1], Yang Yang[1]*

[1]Zhejiang University, [2]FinVolution Group, [3]Fudan University
{fangtr, yangya}@zju.edu.cn, wangchunping02@xinye.com, jiarongxu@fudan.edu.cn

Abstract

Graph Neural Networks (GNNs) are powerful tools for graph representation learning. Despite their rapid development, GNNs also face some challenges, such as *over-fitting*, *over-smoothing*, and *non-robustness*. Previous works indicate that these problems can be alleviated by *random dropping* methods, which integrate augmented data into models by randomly masking parts of the input. However, some open problems of random dropping on GNNs remain to be solved. First, it is challenging to find a universal method that are suitable for all cases considering the divergence of different datasets and models. Second, augmented data introduced to GNNs causes the incomplete coverage of parameters and unstable training process. Third, there is no theoretical analysis on the effectiveness of random dropping methods on GNNs. In this paper, we propose a novel random dropping method called *DropMessage*, which performs dropping operations directly on the propagated messages during the message-passing process. More importantly, we find that DropMessage provides a unified framework for most existing random dropping methods, based on which we give theoretical analysis of their effectiveness. Furthermore, we elaborate the superiority of DropMessage: it stabilizes the training process by reducing sample variance; it keeps information diversity from the perspective of information theory, enabling it become a theoretical upper bound of other methods. To evaluate our proposed method, we conduct experiments that aims for multiple tasks on five public datasets and two industrial datasets with various backbone models. The experimental results show that DropMessage has the advantages of both effectiveness and generalization, and can significantly alleviate the problems mentioned above. Our code is available at: https://github.com/zjunet/DropMessage.

1 Introduction

Graphs, ubiquitous in the real world, are used to present complex relationships among various objects in numerous domains such as social media (social networks), finance (trading networks), and biology (biological networks). As powerful tools for representation learning on graphs, graph neural networks (GNNs) have attracted considerable attention recently (Defferrard, Bresson, and Vandergheynst 2016; Kipf and Welling 2017; Velickovic et al. 2018; Ding, Tang, and

---

*Corresponding author.

Zhang 2018). In particular, GNNs adopt a message-passing schema (Gilmer et al. 2017), in which each node aggregates information from its neighbors in each convolutional layer, and have been widely applied in various downstream tasks such as node classification (Kipf and Welling 2017), link prediction (Kipf and Welling 2016), vertex clustering (Ramaswamy, Gedik, and Liu 2005), and recommendation systems (Ying et al. 2018).

Yet, despite their rapid development, training GNNs on large-scale graphs is facing several challenges such as *over-fitting*, *over-smoothing*, and *non-robustness*. Indeed, compared to other data forms, gathering labels for graph data is expensive and inherently biased, which limits the generalization ability of GNNs due to over-fitting. Besides, representations of different nodes in a GNN tend to become indistinguishable as a result of aggregating information from neighbors recursively. This phenomenon of over-smoothing prevents GNNs from effectively modeling the higher-order dependencies from multi-hop neighbors (Li, Han, and Wu 2018; Xu et al. 2018; Chen et al. 2020; Zhao and Akoglu 2020; Oono and Suzuki 2020, 2019). Recursively aggregating schema makes GNNs vulnerable to the quality of input graphs (Zhu et al. 2019; Zügner, Akbarnejad, and Günnemann 2018). In other words, noisy graphs or adversarial attacks can easily influence a GNN's performance.

The aforementioned problems can be helped by *random dropping* methods (Hinton et al. 2012; Rong et al. 2019; Feng et al. 2020), which integrate augmented data into models by randomly masking parts of the input. These methods (Maaten et al. 2013; Matsuoka 1992; Bishop 1995; Cohen, Rosenfeld, and Kolter 2019) focus on randomly dropping or sampling existing information, and can also be considered as a data augmentation technique. Benefiting from the advantages of being unbiased, adaptive, and free of parameters, random dropping methods have greatly contributed to improving the performance of most GNNs.

However, some open questions related to random dropping methods on GNNs still exist. First, a general and critical issue of existing random dropping methods is that augmented data introduced to GNNs make parameters difficult to converge and the training process unstable. Moreover, it is challenging to find an optimal dropping method suitable to all graphs and models, because different graphs and models are equipped with their own properties and the model performance can be
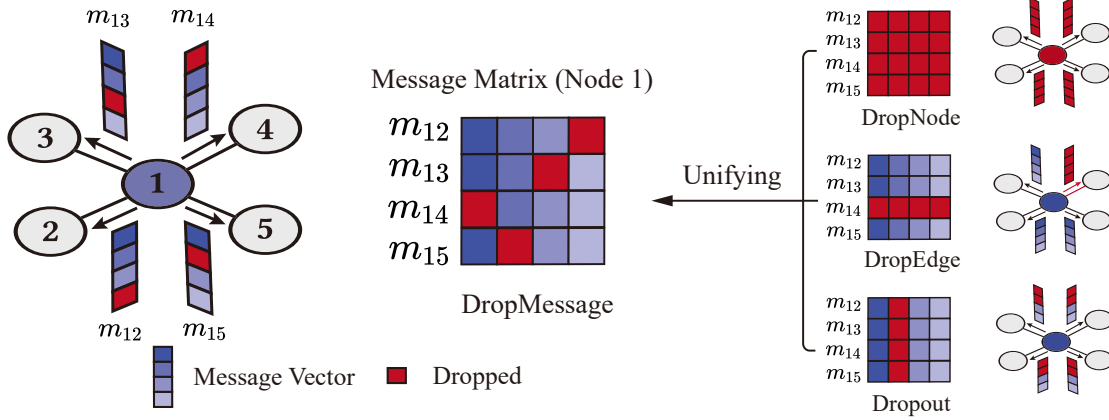
Figure 1: Illustrations of DropMessage and other existing random dropping methods. Considering the messages propagated by the center node (*i.e.*, Node 1), DropMessage allows to propagate distinct messages to different neighbor nodes, and its induced message matrix can be arbitrary. The induced message matrices of other methods obey some explicit constraints and can be regarded as special forms of DropMessage.

influenced greatly by employing various dropping strategies. Furthermore, the answer to how to choose a proper dropping rate when applying these methods is still unclear, and so far no theoretical guarantee has been provided to explain why random dropping methods can improve the performance of a GNN.

In this paper, we propose a novel random dropping method called DropMessage, which can be applied to all message-passing GNNs. As Figure 1 suggests, existing random dropping methods perform dropping on either the node feature matrix (Hinton et al. 2012; Feng et al. 2020) or the adjacency matrix (Rong et al. 2019), while our DropMessage performs dropping operations on the propagated messages, which allows the same node to propagate different messages to its different neighbors. Besides, we unify existing dropping methods into our framework and demonstrate theoretically that conducting random dropping methods on GNNs is equivalent to introducing additional regularization terms to their loss functions, which makes the models more robust. Furthermore, we also elaborate the superiority of our DropMessage whose sample variance is much smaller and training process is more stable. From the perspective of information theory, DropMessage keeps the property of information diversity, and is theoretically regarded as an upper bound of other random dropping methods. To sum up, the contributions of this paper are as follows:

- We propose a novel random dropping method, called DropMessage, for all message-passing GNNs. Existing random dropping methods on GNNs can be unified into our framework via performing masking in accordance with the certain rule on the message matrix. In other words, these methods can be regarded as one special form of DropMessage.

- We theoretically demonstrate the effectiveness of the random dropping methods, filling the gap in this field.

- We conduct sufficient experiments for different downstream tasks, and the experimental results show that

DropMessage can better alleviate over-fitting, over-smoothing, and non-robustness compared to other existing random dropping methods.

2 Related Work

To sort out the key logic of our work, we first review some related work about random dropping methods with a particular focus on GNNs.

In general, random dropping can be regarded as a form of feature-noising schema that alleviates over-fitting by artificially corrupting the training data. As a representative work, Dropout is first introduced by Hinton et al. (Hinton et al. 2012) and has been proved to be effective in many scenarios (Abu-Mostafa 1990; Burges and Schölkopf 1996; Simard et al. 1998; Rifai et al. 2011; Maaten et al. 2013). Besides, Bishop (Bishop 1995) demonstrates the equivalence of corrupted features and L_2-type regularization. Wager et al. (Wager, Wang, and Liang 2013) show that the dropout regularizer is first-order equivalent to an L_2 regularizer that being applied after scaling the features by an estimate of the inverse diagonal Fisher information matrix.

With the rapid development of GNNs, random dropping has also been generalized to the graph field, thus leading to three most common methods: *Dropout* (Hinton et al. 2012), *DropEdge* (Rong et al. 2019) and *DropNode* (Feng et al. 2020). Dropout performs random dropping operation on the node feature matrix, while DropEdge and DropNode, as the name implies, respectively act on the adjacency matrix (edges) and nodes. These random dropping methods can also be regarded as special forms of data augmentation (Shorten and Khoshgoftaar 2019; Frid-Adar et al. 2018; Buslaev et al. 2020; Ding et al. 2022b; Velickovic et al. 2019b), with the advantage of not requiring parameter estimation (Papp et al. 2021; Luo et al. 2021; Chen, Ma, and Xiao 2018; Zeng et al. 2020) and easy to apply. All the methods mentioned above can be used to alleviate *over-fitting* and *over-smoothing* on GNNs. However, they can achieve effective performance only

on some specific datasets and GNNs. The question of how to find an optimal dropping method that suitable for most cases still remains to be explored. Moreover, there is no theoretical explanation about the effectiveness of random dropping methods on GNNs, which adds some ambiguity to the function of these methods.

3   Notations and Preliminaries

Notations. Let $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ represent the graph, where $\mathbf{V} = \{v_1, \ldots, v_n\}$ denotes the set of n nodes, and $\mathbf{E} \subseteq \mathbf{V} \times \mathbf{V}$ is the set of edges between nodes. The node features can be denoted as a matrix $\mathbf{X} = \{x_1, \ldots, x_n\} \in \mathbb{R}^{n \times c}$, where x_i is the feature vector of the node v_i, and c is the dimensionality of node features. The edges describe the relations between nodes and can be represented as an adjacent matrix $\mathbf{A} = \{a_1, \ldots, a_n\} \in \mathbb{R}^{n \times n}$, where a_i denotes the i-th row of the adjacency matrix, and $\mathbf{A}(i, j)$ denotes the relation between nodes v_i and v_j. Also, the node degrees are given by $\mathbf{d} = \{d_1, \ldots, d_n\}$, where d_i computes the sum of edge weights connected to node v_i. Meanwhile, the degree of the whole graph is calculated by $\mathbf{d}(\mathbf{G}) = \sum_i^n d_i$. When we apply message-passing GNNs on $\mathbf{G}$, the message matrix can be represented as $\mathbf{M} = \{m_1, \ldots, m_k\} \in \mathbb{R}^{k \times c'}$, where m_i is a message propagated between nodes, k is the total number of messages propagated on the graph, and c' is the dimension number of the messages.

Message-passing GNNs. Most of the existing GNN models adopt the message-passing framework, where each node sends messages to its neighbors and simultaneously receives messages from its neighbors. In the process of the propagation, node representations are updated based on node feature information and messages from neighbors, which can be formulated as

$$h_i^{(l+1)} = \gamma^{(l)}(h_i^{(l)}, \mathcal{AGG}_{j \in \mathcal{N}(i)}(\phi^{(l)}(h_i^{(l)}, h_j^{(l)}, e_{j,i}))) \quad (1)$$

where $h_i^{(l)}$ denotes the hidden representation of node v_i in the l-th layer, and $\mathcal{N}(i)$ is a set of nodes adjacent to node v_i; $e_{j,i}$ represents the edge from node j to node i; $\phi^{(l)}$ and $\gamma^{(l)}$ are differentiable functions; and $\mathcal{AGG}$ represents the aggregation operation. From the perspective of the message-passing schema, we can gather all the propagated messages into a message matrix $\mathbf{M} \in \mathbb{R}^{k \times c'}$. Specifically, each row of the message matrix $\mathbf{M}$ corresponds to a message propagated on a directed edge, which can be expressed as below:

$$\mathbf{M}_{(i,j)}^{(l)} = \phi^{(l)}(h_i^{(l)}, h_j^{(l)}, e_{j,i}^{(l)})$$

where ϕ denotes the mapping that generates the messages, c' is the dimension number of the messages, and the row number k of the message matrix $\mathbf{M}$ is equal to the directed edge number in the graph.

4   Our Approach

In this section, we introduce our proposed *DropMessage*, which can be applied to all message-passing GNNs. We first describe the details of our approach, and further prove that

the most common existing random dropping methods, i.e., Dropout, DropEdge and DropNode, can be unified into our framework. Based on that, we give a theoretical explanation of the effectiveness of these methods. After that, we theoretically analyze the superiority of DropMessage in terms of stabilizing the training process and keeping information diversity. Finally, we derive a theoretical upper bound to guide the selection of dropping rate δ.

4.1   DropMessage

Algorithm description. Different from existing random dropping methods, DropMessage performs directly on the message matrix $\mathbf{M}$ instead of the feature matrix or the adjacency matrix. More specifically, DropMessage conducts dropping on the message matrix with the dropping rate δ, which means that $\delta|\mathbf{M}|$ elements of the message matrix will be masked in expectation. Formally, this operation can be regarded as a sampling process. For each element $\mathbf{M}_{i,j}$ in the message matrix, we generate an independent mask $\epsilon_{i,j}$ to determine whether it will be preserved or not, according to a Bernoulli distribution $\epsilon_{i,j} \sim Bernoulli(1 - \delta)$. Then, we obtain the perturbed message matrix $\widetilde{\mathbf{M}}$ by multiplying each element with its mask. Finally, we scale $\widetilde{\mathbf{M}}$ with the factor of $\frac{1}{1-\delta}$ to guarantee that the perturbed message matrix is equal to the original message matrix in expectation. Thus, the whole process can be expressed as $\widetilde{\mathbf{M}}_{i,j} = \frac{1}{1-\delta}\epsilon_{i,j}\mathbf{M}_{i,j}$, where $\epsilon_{i,j} \sim Bernoulli(1 - \delta)$. The applied GNN model then propagates information via the perturbed message matrix $\widetilde{\mathbf{M}}$ instead of the original message matrix. It should be noted that DropMessage only affects on the training process.

Practical implementation. In practice, we do NOT need to generate the complete message matrix explicitly, because each row in the message matrix represents a distinct directed edge in the graph, and our proposed DropMessage can be applied to every directed edge independently. This property allows DropMessage to be easily parallelized, *e.g.*, edge-wise, node-wise, or batch-wise, and to be applied to the message-passing backbone model *without increasing time or space complexity*.

Unifying random dropping methods. As we have mentioned above, DropMessage differs from existing methods by directly performing on messages instead of graphs. However, in intuition, the dropping of features, edges, nodes or messages will all eventually act on the message matrix. It inspires us to explore the theoretical connection between different dropping methods. As a start, we demonstrate that Dropout, DropEdge, DropNode, and DropMessage can all be formulated as Bernoulli sampling processes in Table 1. More importantly, we find that existing random dropping methods are actually special cases of DropMessage, and thus can be expressed in a uniform framework.

Lemma 1. *Dropout, DropEdge, DropNode, and DropMessage perform random masking on the message matrices in accordance with certain rules.*

We provide the equivalent operation on the message matrix of each method below.

Table 1: Overview of different random dropping methods in a view of Bernoulli sampling process.

Method	Formula
Dropout	$\widetilde{\mathbf{X}}_{i,j} = \epsilon \mathbf{X}_{i,j}$
DropEdge	$\widetilde{\mathbf{A}}_{i,j} = \epsilon \mathbf{A}_{i,j}$
DropNode	$\widetilde{\mathbf{X}}_i = \epsilon \mathbf{X}_i$
DropMessage	$\widetilde{\mathbf{M}}_{i,j} = \epsilon \mathbf{M}_{i,j}$

$s.t.\ \epsilon \sim Bernoulli(1 - \delta)$

Dropout. Dropping the elements $X_{drop} = \{\mathbf{X}_{i,j} | \epsilon_{i,j} = 0\}$ in the feature matrix $\mathbf{X}$ is equivalent to masking elements $M_{drop} = \{\mathbf{M}_{i,j} | source(\mathbf{M}_{i,j}) \in X_{drop}\}$ in the message matrix $\mathbf{M}$, where $source(\mathbf{M}_{i,j})$ indicates which element in the feature matrix that $\mathbf{M}_{i,j}$ corresponds to.

DropEdge. Dropping the elements $E_{drop} = \{\mathbf{E}_{i,j} | \mathbf{A}_{i,j} = 1\ and\ \epsilon_{i,j} = 0\}$ in the adjacency matrix $\mathbf{A}$ is equivalent to masking elements $M_{drop} = \{\mathbf{M}_i | edge(\mathbf{M}_i) \in E_{drop}\}$ in the message matrix $\mathbf{M}$, where $edge(\mathbf{M}_i)$ indicates which edge that $\mathbf{M}_i$ corresponds to.

DropNode. Dropping the elements $V_{drop} = \{\mathbf{X}_i | \epsilon_i = 0\}$ in the feature matrix $\mathbf{X}$ is equivalent to masking elements $M_{drop} = \{\mathbf{M}_i | node(\mathbf{M}_i) \in V_{drop}\}$ in the message matrix $\mathbf{M}$, where $node(\mathbf{M}_i)$ indicates which row in the feature matrix that $\mathbf{M}_i$ corresponds to.

DropMessage. This method directly performs random masking on the message matrix $\mathbf{M}$.

According to above descriptions, we find DropMessage conduct finest-grained masking on the message matrix, which makes it the most flexible dropping method, and other methods can be regarded as a special form of DropMessage.

Theoretical explanation of effectiveness. Previous studies have explored and explained why random dropping works in the filed of computer vision (Wager, Wang, and Liang 2013; Wan et al. 2013). However, to the best of our knowledge, the effectiveness of random dropping on GNNs has not been studied yet. To fill this gap, based on the unified framework of existing methods, we next provide a theoretical analysis.

Theorem 1. *Unbiased random dropping on GNNs methods introduce an additional regularization term into the objective functions, which makes the models more robust.*

Proof. For analytical simplicity, we assume that the downstream task is a binary classification and we apply a single layer GCN (Kipf and Welling 2017) as the backbone model, which can be formulated as $\mathbf{H} = \overline{\mathbf{B}}\mathbf{M}\mathbf{W}$, where $\mathbf{M}$ denotes the message matrix, $\mathbf{W}$ denotes the transformation matrix, $\mathbf{B} \in \mathbb{R}^{n \times k}$ indicates which messages should be aggregated by each node and $\overline{\mathbf{B}}$ is its normalized form. Also, we adopt sigmoid as non-linear function and present the result as $\mathbf{Z} = sigmoid(\mathbf{H})$. When we use cross-entropy as loss function, the objective function can be expressed as follows:

$$L_{CE} = \sum_{j,y_j=1} log(1 + e^{-h_j}) + \sum_{k,y_k=0} log(1 + e^{h_k}) \quad (2)$$

When performing random dropping on graphs, we use the perturbed message matrix $\widetilde{\mathbf{M}}$ instead of the original message matrix $\mathbf{M}$. Thus, the objective function in expectation can be expressed as follows:

calculate it with probability& linear algebra

$$E(\widetilde{L}_{CE}) = L_{CE} + \sum_i \frac{1}{2} z_i(1 - z_i)Var(\tilde{h}_i) \quad (3)$$

More details of the derivation can be found in Appendix. As shown in Equation 3, random dropping methods on graphs introduce an extra regularization to the objective function. For binary classification tasks, this regularization enforces the classification probability approach to 0 or 1, thus a clearer judgment can be obtained. By reducing the variance of $\tilde{h}_i$, random dropping methods motivate the model to extract more essential high-level representations. Therefore, the robustness of the models is enhanced. It is noted that Equation 3 can be well generalized to multi-classification tasks by extending dimension of the model output. Formally, when dealing with the multi-classification task, the final objective function can be expressed as $E(\widetilde{L}_{CE}) = L_{CE} + \sum_i \frac{1}{2} z_i^{c_i}(1 - z_i^{c_i})Var(\tilde{h}_i^{c_i})$, where c_i is the label of node v_i, and the superscript indicates which dimension of the vector is selected. $\square$

4.2 Advantages of DropMessage

We give two additional analysis to demonstrate the advantages of DropMessage on two aspects: stabilizing the training process and keeping diverse information.

Reducing sample variance. All random dropping methods are challenged by the problem of unstable training process. As existing works suggest, it is caused by the random noises introduced into each training epoch. These noises then add the difficulty of parameter coverage and the unstability of training process. Generally, *sample variance* can be used to measure the degree of stability. According to Table 1, the input of each training epoch can be regarded as a random sample of the whole graph, and the sample variance is calculated by the average difference of every two independent samples. Compared with other random dropping methods, DropMessage effectively alleviates the aforementioned problem by reducing the sample variance.

cause and measure of unstability

Theorem 2. *DropMessage presents the smallest sample variance among existing random dropping methods on message-passing GNNs with the same dropping rate δ.*

We leave the proof in Appendix. Intuitively, DropMessage independently determines whether an element in the message matrix is masked or not, which is exactly the smallest Bernoulli trail for random dropping on the message matrix. By reducing the sample variance, DropMessage diminishes the difference of message matrices among distinct training epochs, which stabilizes the training process and expedites the convergence. The reason why DropMessage has the minimum sample variance is that it is the finest-grained random dropping method for GNN models. When applying DropMessage, each element $\mathbf{M}_{i,j}$ will be independently judged that whether it should be masked.

$$\sum_{i=1}^n D(x_1) \leq D(nx_1) = n^2 D(x_1)$$

Keeping diverse information. In the following, we compare different random dropping methods with their degree of losing *information diversity*, from the perspective of information theory.

Definition 1. *The information diversity consists of feature diversity and topology diversity. We define feature diversity as $FD_G = card(\{\|M_{SN(v_i),l}\|_0 \geq 1\})$, where $v_i \in \mathbf{V}$, $l \in [0, c)$, $SN(v_i)$ indicates the slice of the row numbers corresponding to the edges sourced from v_i; topology diversity is defined as $TD_G = card(\{\|M_j\|_0 \geq 1\})$, where $j \in [0, k)$. $M \in \mathbb{R}^{k \times c}$ represents the message matrix, $\|\cdot\|_0$ calculates the zero norm of the input vector, and $card(\cdot)$ counts the number of elements in the set.*

In other words, feature diversity is defined as the total number of preserved feature dimensions from distinct source nodes; topology diversity is defined as the total number of directed edges propagating at least one dimension message. With the above definition, we claim that a method possesses the ability of *keeping information diversity* only under the condition where neither the feature diversity nor the topology diversity decreases after random dropping.

Lemma 2. *None of Dropout, DropEdge, and DropNode is able to keep information diversity.*

According to Definition 1, when we drop an element of the feature matrix $\mathbf{X}$, all corresponding elements in the message matrix are masked and the feature diversity is decreased by 1. When we drop an edge in adjacency matrix, the corresponding two rows for undirected graphs in the message matrix are masked and the topology diversity is decreased by 2. Similarly, when we drop a node, *i.e.*, a row in the feature matrix, elements in the corresponding rows of the message matrix are all masked. Both the feature diversity and the topology diversity are therefore decreased. Thus, for all of these methods, their feature and topology information cannot be completely recovered by propagated messages, leading to the loss of information diversity.

Theorem 3. *DropMessage can keep information diversity in expectation when $\delta_i \leq 1 - min(\frac{1}{d_i}, \frac{1}{c})$, where δ_i is the dropping rate for node v_i, d_i is the out-degree of v_i, and c is the feature dimension.*

Proof. DropMessage conducts random dropping directly on message matrix $\mathbf{M}$. To keep the diversity of the topology information, we expect that at least one element of each row in message matrix $\mathbf{M}$ can be preserved in expectation:

$$E(|\mathbf{M}_f|) \geq 1 \Rightarrow (1 - \delta)c \geq 1 \Rightarrow \delta \leq 1 - \frac{1}{c} \qquad (4)$$

To keep the diversity of the feature information, we expect that for every element in the feature matrix $\mathbf{X}$, at least one of its corresponding elements in the message matrix $\mathbf{M}$ is preserved in expectation:

$$E(|\mathbf{M}_e|) \geq 1 \Rightarrow (1 - \delta_i)d_i \geq 1 \Rightarrow \delta_i \leq 1 - \frac{1}{d_i} \qquad (5)$$

Therefore, to keep the information diversity, the dropping rate δ_i should satisfy both Equation 4 and Equation 5 as

$$\delta_i \leq 1 - min(\frac{1}{d_i}, \frac{1}{c}) \qquad (6)$$

From the perspective of information theory, a random dropping method with the capability of keeping information diversity can preserve more information and theoretically perform better than those without such capability. Thus, it can explain why our method performs better than those existing dropping methods. Actually, we may only set one dropping rate δ for the whole graph rather than for each node in practice. Consequently, both DropMessage and other methods may lose some information. However, DropMessage still preserves more information than other methods with the same dropping rate even under this circumstance. It is demonstrated that DropMessage remains its advantage in real-world scenarios.

5   Experiments

5.1   Experimental Setup

We empirically validate the effectiveness and adaptability of our proposed DropMessage in this section. In particular, we explore the following questions: 1) Does DropMessage outperform other random dropping methods on GNNs? 2) Could DropMessage further improve the robustness and training efficiency of GNNs? 3) Does information diversity (described in Definition 1) matter in GNNs?

Datasets. We employ 7 graph datasets in our experiments, including 5 public datasets *Cora, CiteSeer, PubMed, ogbn-arxiv, Flickr* and 2 industrial datasets *FinV, Telecom*.

- *Cora, CiteSeer, PubMed, ogbn-arxiv:* These 4 different citation networks are widely used as graph benchmarks (Sen et al. 2008; Hu et al. 2020). We conduct node classification tasks on each dataset to determine the research area of papers/researchers. We also consider link prediction on the first three graphs to predict whether one paper cites another.

- *Flickr:* It is provided by Flickr, the largest photo-sharing website (Zeng et al. 2020). One node in the graph represents one image uploaded to Flickr. If two images share some common properties (*e.g.*, same geographic location, same gallery, or comments by the same user), an edge between the nodes of these two images will appear. We conduct the node classification task that aims to categorize these images into 7 classes determined by their tags.

- *FinV, Telecom:* These are two real-world mobile communication networks provided by FinVolution Group (Yang et al. 2019) and China Telecom (Yang et al. 2021), respectively. In the two datasets, nodes represent users, and edges indicate the situation where two users have communicated with each other at a certain frequency. The task is to identify whether a user is a default borrower or a telecom fraudster.

Baseline methods. We compare our proposed DropMessage with other existing random dropping methods, including Dropout (Hinton et al. 2012), DropEdge (Rong et al. 2019), and DropNode (Feng et al. 2020). We adopt these dropping methods on various GNNs as the backbone model, and compare their performances on different datasets.

Table 2: Comparison results of different random dropping methods. The best results are in bold, while the second-best ones are underlined.

Task & Dataset / Model	Node classification				Flickr	Telecom	FinV	Link prediction		
	Cora	CiteSeer	PubMed	ogbn-arxiv				Cora	CiteSeer	PubMed
GCN	80.68	70.83	78.97	70.08	0.5188	0.6080	0.4220	0.9198	0.8959	0.9712
GCN-Dropout	<u>83.16</u>	71.48	<u>79.13</u>	<u>71.16</u>	<u>0.5222</u>	0.6601	0.4526	0.9278	**0.9107**	<u>0.9766</u>
GCN-DropEdge	81.69	71.43	79.06	70.88	0.5214	<u>0.6650</u>	<u>0.4729</u>	<u>0.9295</u>	0.9067	0.9762
GCN-DropNode	83.04	**72.12**	79.00	70.98	0.5213	0.6243	0.4571	0.9238	0.9052	0.9748
GCN-DropMessage	**83.33**	<u>71.83</u>	**79.20**	**71.27**	**0.5223**	**0.6710**	**0.4876**	**0.9305**	<u>0.9071</u>	**0.9772**
GAT	81.35	70.14	77.20	70.32	0.4988	0.7050	0.4467	0.9118	0.8895	0.9464
GAT-Dropout	**82.41**	71.31	**78.31**	**71.28**	0.4998	0.7382	0.4539	0.9182	0.9055	0.9536
GAT-DropEdge	81.82	71.17	77.70	70.67	<u>0.5004</u>	<u>0.7568</u>	**0.4896**	0.9206	0.9037	0.9493
GAT-DropNode	82.08	<u>71.44</u>	77.98	70.96	0.4992	0.7214	0.4647	**0.9224**	**0.9104**	**0.9566**
GAT-DropMessage	<u>82.20</u>	71.48	<u>78.14</u>	<u>71.13</u>	**0.5013**	**0.7574**	<u>0.4861</u>	<u>0.9216</u>	<u>0.9076</u>	<u>0.9553</u>
APPNP	81.45	70.62	79.79	69.11	0.5047	0.6217	0.3952	0.9058	0.8844	0.9531
APPNP-Dropout	82.23	71.93	<u>79.92</u>	<u>69.36</u>	0.5055	0.6578	0.4023	0.9119	0.9071	0.9611
APPNP-DropEdge	**82.75**	<u>72.10</u>	79.83	69.15	<u>0.5061</u>	<u>0.6591</u>	0.4149	<u>0.9139</u>	<u>0.9131</u>	<u>0.9626</u>
APPNP-DropNode	81.79	71.50	79.81	69.27	0.5053	0.6412	<u>0.4182</u>	0.9068	0.8979	0.9561
APPNP-DropMessage	<u>82.37</u>	**72.65**	**80.04**	**69.72**	**0.5072**	0.6619	**0.4378**	**0.9165**	**0.9141**	**0.9634**

Backbone models. In this paper, we mainly consider three mainstream GNNs as our backbone models: GCN (Kipf and Welling 2017), GAT (Velickovic et al. 2018), and APPNP (Klicpera, Bojchevski, and Günnemann 2019). We take the official practice of these methods while make some minor modifications. All these backbone models have random dropping modules for different steps in their model implementation. For instance, GAT models perform random dropping after self-attention calculation, while APPNP models perform random dropping at the beginning of each iteration. For a fair comparison, we unify the implementation of random dropping modules in the same step for different backbone models. We fix Dropout, DropEdge, and DropNode on the initial input and fix DropMessage at the start point of the message propagation process.

5.2 Comparison Results

Table 2 summarizes the overall results. For the node classification task, the performance is measured by accuracy on four public datasets (Cora, CiteSeer, PubMed, ogbn-arxiv). As for Flickr and two imbalanced industrial datasets, we employ F1 scores. When it comes to the link prediction task, we calculate the AUC values for comparisons. Considering the space limitation, the std values of the experimental results are presented in the Appendix.

Effect of random dropping methods. It is observed that random dropping methods consistently outperform GNNs without random dropping in both node classification and link prediction. Besides, we see that the effects of random dropping methods vary over different datasets, backbone models, and downstream tasks. For example, random dropping methods on APPAP obtain an average accuracy improvement of 1.4% on CiteSeer, while 0.1% on PubMed. Meanwhile, random dropping methods achieve 2.1% accuracy improvement for GCN on Cora, while only 0.8% for GAT.

Comparison of different dropping methods. Our pro-

posed DropMessage works well in all settings, exhibiting its strong adaptability to various scenarios. Overall, we have 21 settings under the node classification task, each of which is a combination of different backbone models and datasets (*e.g.*, GCN-Cora). It is showed that DropMessage achieves the optimal results in 15 settings, and gets sub-optimal results in the rest. As to 9 setttings under the link prediction task, DropMessage achieves the optimal results in 5 settings, and sub-optimal results in the rest. Moreover, the stable performance of DropMessage over all datasets compared to other methods is clearly presented. Taking DropEdge as the counterexample, it appears strong performance on industrial datasets but demonstrates a clear drop on public ones. A reasonable explanation is that the message matrix patterns reserved by distinct mask methods vary from each other as presented in Table 1. With the favor of its finest-grained dropping strategy, DropMessage obtains smaller inductive bias. Thus, compared with other methods, DropMessage is more applicable in most scenarios.

5.3 Additional Results

Robustness analysis. We study the robustness of dropping methods through measuring their ability of handling perturbed graphs. To guarantee that the initial data is comparatively clean, we conduct experiments on three citation networks: Cora, CiteSeer, and PubMed. We randomly add a certain ratio of edges into these datasets and perform the node classification. We find that all the random dropping methods have positive effects when the perturbation rate increases from 0% to 30%. The average improvement in the case of 30% perturbation reached 37% compared to that without perturbation, which indicates that the random dropping methods strengthen the robustness of GNN models. Besides, our proposed DropMessage shows its versatility and outperforms other dropping methods in noisy situations. Detailed results are exhibited in Appendix.
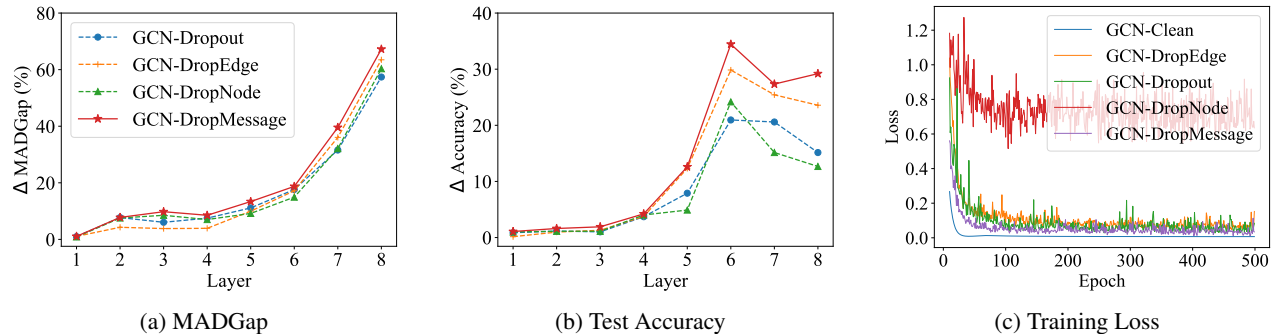
| | (a) MADGap | (b) Test Accuracy | (c) Training Loss |

Figure 2: Over-Smoothing and Training Process Analysis.

Over-smoothing analysis. Over-smoothing is a common issue on GNNs (Li, Han, and Wu 2018), which implies that the node representations become indistinguishable as the network depth increases. In this part, we evaluate the effects that various random dropping methods exert on this issue, and measure the degree of over-smoothing by MADGap (Chen et al. 2020). It should be noted that here a smaller value indicates the more indistinguishable node representations and vice versa. Experiments are conducted on Cora with GCNs serving as backbone models. Figure 2a and Figure 2b show the relative increase of MADGap values and test accuracies of the final node representations compared to the original model without any random dropping techniques. The results indicate that all these random dropping methods can alleviate over-smoothing by increasing the MADGap values and test accuracies when the depth of the model increases. Among all random dropping methods, our proposed DropMessage exhibits a superiority of consistency. It obtains an average improvement of 3.3% on MADGap values and an average improvement of 4.9% on test accuracies compared to other random dropping methods when the layer number $l \geq 3$. This result can be explained by the fact that DropMessage can generate more various messages than other methods, which prevents the nodes from converging to the same representations to some extent. A more detailed theoretical explanation can be found in Appendix.

Training process analysis. We conduct experiments to analyze the loss during the training process when employing different random dropping methods. Figure 2c shows the change of loss in GCN training processes when employing different random dropping methods on Cora. Furthermore, the similar training loss curves can be drawn under other experimental settings. The experimental results suggest that DropMessage presents the smallest sample variance among all methods, thus achieving the fastest convergence and the most stable performance. This is consistent with the theoretical results in Section 4.2.

Information diversity analysis. We conduct experiments to evaluate the importance of information diversity for message-passing GNN models. We set Cora as our experimental dataset, which contains 2708 nodes and 5429 edges. The average node degree of Cora is close to 4. According to Equation 6, the upper bound of dropping rate is calculated from

Table 3: Classification accuracy (%) for information diversity analysis (where AVG denotes average, and NW denotes nodewise).

Model	GCN		GAT		APPNP	
	AVG	NW	AVG	NW	AVG	NW
Accuracy (%)	81.62	82.67	80.81	81.61	80.71	81.56

the node degree and the feature dimension. The feature dimension number of Cora is 1433, which is much larger than the number of node degree. Therefore, the upper bound is only determined by the degree of the node. In (backbone)-nodewise settings, we set the dropping rate to be equal to its upper bound $\delta_i = 1 - \frac{1}{d_i}$ for each node. In (backbone)-average settings, we set the dropping rate $\delta_i = 0.75 + \epsilon_i$, where $\epsilon_i \sim Uniform(-0.15, 0.15)$. Both of these settings employ DropMessage. The average random dropping rate of all nodes is almost identical under these two settings, but only the former one can keep the information diversity in expectation. Table 3 presents the results. The (backbone)-nodewise settings outperform (backbone)-average settings regardless of which backbone model is selected.

6 Conclusion

In this paper, we propose DropMessage, a general random dropping method for message-passing GNN models. We first unify all random dropping methods to our framework via performing dropping on the message matrix and analyzing their effects. Then we illustrate the superiority of DropMessage theoretically in stabilizing the training process and keeping information diversity. Due to its fine-grained dropping operations on the message matrix, DropMessage shows greater applicability in most cases. By conducting experiments for multiple tasks on five public datasets and two industrial datasets, we demonstrate the effectiveness and generalization of our proposed method.

7 Acknowledgments

References

Abu-Mostafa, Y. S. 1990. Learning from hints in neural networks. *Journal of complexity*, 6(2): 192–198.

Bishop, C. M. 1995. Training with Noise is Equivalent to Tikhonov Regularization. *Neural Computation*, 7: 108–116.

Burges, C.; and Schölkopf, B. 1996. Improving the Accuracy and Speed of Support Vector Machines. In *NIPS*.

Buslaev, A. V.; Parinov, A.; Khvedchenya, E.; Iglovikov, V. I.; and Kalinin, A. A. 2020. Albumentations: fast and flexible image augmentations. *ArXiv*, abs/1809.06839.

Cai, C.; Wang, D.; and Wang, Y. 2021. Graph coarsening with neural networks. *arXiv preprint arXiv:2102.01350*.

Chen, D.; Lin, Y.; Li, W.; Li, P.; Zhou, J.; and Sun, X. 2020. Measuring and Relieving the Over-smoothing Problem for Graph Neural Networks from the Topological View. In *AAAI*.

Chen, J.; Ma, T.; and Xiao, C. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. *ICLR*.

Cohen, J. M.; Rosenfeld, E.; and Kolter, J. Z. 2019. Certified Adversarial Robustness via Randomized Smoothing. In *ICML*.

Dai, E.; Aggarwal, C.; and Wang, S. 2021. Nrgnn: Learning a label noise resistant graph neural network on sparsely and noisily labeled graphs. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 227–236.

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional neural networks on graphs with fast localized spectral filtering.

Ding, K.; Wang, J.; Caverlee, J.; and Liu, H. 2022a. Meta propagation networks for graph few-shot semi-supervised learning. AAAI.

Ding, K.; Xu, Z.; Tong, H.; and Liu, H. 2022b. Data Augmentation for Deep Graph Learning: A Survey. *ArXiv*, abs/2202.08235.

Ding, K.; Xu, Z.; Tong, H.; and Liu, H. 2022c. Data augmentation for deep graph learning: A survey. *arXiv preprint arXiv:2202.08235*.

Ding, M.; Tang, J.; and Zhang, J. 2018. Semi-supervised learning on graphs with generative adversarial nets. In *CIKM*, 913–922.

Elinas, P.; and Bonilla, E. V. 2022. Addressing Over-Smoothing in Graph Neural Networks via Deep Supervision. *ArXiv*, abs/2202.12508.

Feng, W.; Zhang, J.; Dong, Y.; Han, Y.; Luan, H.; Xu, Q.; Yang, Q.; Kharlamov, E.; and Tang, J. 2020. Graph Random Neural Networks for Semi-Supervised Learning on Graphs. *NeurIPS*, 33.

Frid-Adar, M.; Diamant, I.; Klang, E.; Amitai, M. M.; Goldberger, J.; and Greenspan, H. 2018. GAN-based Synthetic Medical Image Augmentation for increased CNN Performance in Liver Lesion Classification. *Neurocomputing*, 321: 321–331.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural message passing for quantum chemistry. In *ICML*, 1263–1272.

Hinton, G. E.; Srivastava, N.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *ArXiv*, abs/1207.0580.

Hu, W.; Fey, M.; Zitnik, M.; Dong, Y.; Ren, H.; Liu, B.; Catasta, M.; and Leskovec, J. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *arXiv preprint arXiv:2005.00687*.

Jin, W.; Zhao, L.; Zhang, S.; Liu, Y.; Tang, J.; and Shah, N. 2021. Graph condensation for graph neural networks. *arXiv preprint arXiv:2110.07580*.

Kipf, T. N.; and Welling, M. 2016. Variational graph auto-encoders. In *ArXiv*, volume abs/1611.07308.

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 *ICLR*.

Li, Q.; Han, Z.; and Wu, X.-M. 2018. Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning. In *AAAI*.

Liu, M.; Gao, H.; and Ji, S. 2020. Towards Deeper Graph Neural Networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM.

Luo, D.; Cheng, W.; Yu, W.; Zong, B.; Ni, J.; Chen, H.; and Zhang, X. 2021. Learning to Drop: Robust Graph Neural Network via Topological Denoising. *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*.

Maaten, L. V. D.; Chen, M.; Tyree, S.; and Weinberger, K. Q. 2013. Learning with Marginalized Corrupted Features. In *ICML*.

Matsuoka, K. 1992. Noise injection into inputs in back-propagation learning. *IEEE Trans. Syst. Man Cybern.*, 22: 436–440.

Ming Chen, Z. W.; Zengfeng Huang, B. D.; and Li, Y. 2020. Simple and Deep Graph Convolutional Networks.

Oono, K.; and Suzuki, T. 2019. On Asymptotic Behaviors of Graph CNNs from Dynamical Systems Perspective. *ArXiv*, abs/1905.10947.

Oono, K.; and Suzuki, T. 2020. Graph Neural Networks Exponentially Lose Expressive Power for Node Classification. *arXiv: Learning*.

Papp, P. A.; Martinkus, K.; Faber, L.; and Wattenhofer, R. 2021. DropGNN: Random Dropouts Increase the Expressiveness of Graph Neural Networks. In *NeurIPS*.

Ramaswamy, L.; Gedik, B.; and Liu, L. 2005. A distributed approach to node clustering in decentralized peer-to-peer networks. In *TPDS*, volume 16, 814–829.

Rifai, S.; Dauphin, Y.; Vincent, P.; Bengio, Y.; and Muller, X. 2011. The Manifold Tangent Classifier. In *NIPS*.

Rong, Y.; Huang, W.; Xu, T.; and Huang, J. 2019. Dropedge: Towards deep graph convolutional networks on node classification. In *ICLR*.

Sen, P.; Namata, G.; Bilgic, M.; Getoor, L.; Galligher, B.; and Eliassi-Rad, T. 2008. Collective classification in network data. volume 29, 93–93.

Shannon, C. E. 2001. A mathematical theory of communication. *Bell Syst. Tech. J.*, 27: 623–656.

Shorten, C.; and Khoshgoftaar, T. M. 2019. A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, 6: 1–48.

Simard, P. Y.; LeCun, Y. A.; Denker, J. S.; and Victorri, B. 1998. Transformation invariance in pattern recognition—tangent distance and tangent propagation. In *Neural networks: tricks of the trade*, 239–274. Springer.

Sun, K.; Lin, Z.; and Zhu, Z. 2020. Multi-stage self-supervised learning for graph convolutional networks on graphs with few labeled nodes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 5892–5899.

Velickovic, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio', P.; and Bengio, Y. 2018. Graph Attention Networks. In *International Conference on Learning Representations*.

Velickovic, P.; Fedus, W.; Hamilton, W. L.; Liò, P.; Bengio, Y.; and Hjelm, R. D. 2019a. Deep Graph Infomax. *ICLR (Poster)*, 2(3): 4.

Velickovic, P.; Fedus, W.; Hamilton, W. L.; Lio', P.; Bengio, Y.; and Hjelm, R. D. 2019b. Deep Graph Infomax. *ICLR*.

Verma, V.; Lamb, A.; Beckham, C.; Najafi, A.; Mitliagkas, I.; Lopez-Paz, D.; and Bengio, Y. 2019. Manifold mixup: Better representations by interpolating hidden states. In *International Conference on Machine Learning*, 6438–6447. PMLR.

Wager, S.; Wang, S. I.; and Liang, P. 2013. Dropout Training as Adaptive Regularization. In *NIPS*.

Wan, L.; Zeiler, M. D.; Zhang, S.; LeCun, Y.; and Fergus, R. 2013. Regularization of Neural Networks using DropConnect. In *ICML*.

Wu, Z.; Ramsundar, B.; Feinberg, E. N.; Gomes, J.; Geniesse, C.; Pappu, A. S.; Leswing, K.; and Pande, V. S. 2017. MoleculeNet: A Benchmark for Molecular Machine Learning. *arXiv: Learning*.

Xu, K.; Chen, H.; Liu, S.; Chen, P.-Y.; Weng, T.-W.; Hong, M.; and Lin, X. 2019. Topology Attack and Defense for Graph Neural Networks: An Optimization Perspective. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Xu, K.; Li, C.; Tian, Y.; Sonobe, T.; Kawarabayashi, K.; and Jegelka, S. 2018. Representation Learning on Graphs with Jumping Knowledge Networks. In *ICML*.

Yang, Y.; Xu, Y.; Sun, Y.; Dong, Y.; Wu, F.; and Zhuang, Y. 2021. Mining Fraudsters and Fraudulent Strategies in Large-Scale Mobile Social Networks. *IEEE Transactions on Knowledge and Data Engineering*, 33: 169–179.

Yang, Y.; Xu, Y.; Wang, C.; Sun, Y.; Wu, F.; Zhuang, Y.; and Gu, M. 2019. Understanding Default Behavior in Online Lending. *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*.

Ying, R.; He, R.; Chen, K.; Eksombatchai, P.; Hamilton, W. L.; and Leskovec, J. 2018. Graph convolutional neural networks for web-scale recommender systems. In *SIGKDD*, 974–983.

Zeng, H.; Zhou, H.; Srivastava, A.; Kannan, R.; and Prasanna, V. 2020. GraphSAINT: Graph Sampling Based Inductive Learning Method. *ArXiv*, abs/1907.04931.

Zhang, H.; Cisse, M.; Dauphin, Y. N.; and Lopez-Paz, D. 2017. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*.

Zhao, L.; and Akoglu, L. 2020. PairNorm: Tackling Over-smoothing in GNNs. *ArXiv*, abs/1909.12223.

Zhao, T.; Liu, Y.; Neves, L.; Woodford, O.; Jiang, M.; and Shah, N. 2021. Data augmentation for graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 11015–11023.

Zhu, D.; Zhang, Z.; Cui, P.; and Zhu, W. 2019. Robust graph convolutional networks against adversarial attacks. In *SIGKDD*, 1399–1407.

Zügner, D.; Akbarnejad, A.; and Günnemann, S. 2018. Adversarial Attacks on Neural Networks for Graph Data. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.

8 Appendix

8.1 Derivation Details

Detailed proof of Theorem 1.

Theorem 1 *Unbiased random dropping methods introduce an extra regularization term into the objective functions, which make the models more robust.*

We give more derivation details of Theorem 1. When we use cross-entropy as the loss function, the objective function can be expressed as follows:

$$
L_{CE} = - \sum_{j,y_j=1} log(z_j) - \sum_{k,y_k=0} log(1 - z_k)
$$
$$
= \sum_{j,y_j=1} log(1 + e^{-h_j}) + \sum_{k,y_k=0} log(1 + e^{h_k})
$$

According to above equation, the initial objective function is $L_{CE} = \sum_{j,y_j=1} log(1 + e^{-h_j}) + \sum_{k,y_k=0} log(1 + e^{h_k})$. When we perturb the message matrix, the objective function can be regarded as a process of adding a bias to the original function, expressed as follows:

$$
E(\widetilde{L}_{CE}) = \sum_{j,y_j=1} [log(1 + e^{-h_j}) + E(f(\tilde{h}_j, h_j))]
$$
$$
+ \sum_{k,y_k=0} [log(1 + e^{h_k}) + E(g(\tilde{h}_k, h_k))]
$$

where $f(\tilde{h}_j, h_j) = log(1 + e^{-\tilde{h}_j}) - log(1 + e^{-h_j})$, and $g(\tilde{h}_k, h_k) = log(1 + e^{\tilde{h}_k}) - log(1 + e^{h_k})$. We can approximate it with the second-order Taylor expansion of $f(.)$ and $g(.)$ around h_i. Thus, the objective function in expectation can be expressed as bellow:

$$
E(\widetilde{L}_{CE}) = L_{CE}
$$
$$
+ E(\sum_{j,y_j=1} [(-1 + z_j)(\tilde{h}_j - h_j) + \frac{1}{2}z_j(1 - z_j)(\tilde{h}_j - h_j)^2])
$$
$$
+ E(\sum_{k,y_k=0} [z_k(\tilde{h}_k - h_k) + \frac{1}{2}z_k(1 - z_k)(\tilde{h}_k - h_k)^2])
$$
$$
= L_{CE} + \sum_i \frac{1}{2}z_i(1 - z_i)Var(\tilde{h}_i)
$$

Proof of Theorem 2.

Theorem 2 *DropMessage presents the smallest sample variance among all existing random dropping methods on message-passing GNNs with the same dropping rate δ.*

Proof. As stated in Lemma 1, all random dropping methods on graphs can be converted to masking operations on the message matrix $\mathbf{M}$. We can measure the difference of message matrices in different epochs by the way of comparing the sample variance of random dropping methods, which can be measured via the norm variance of the message matrix $|\mathbf{M}|_F$. Without loss of generality, we assume the original message matrix $\mathbf{M}$ is $\mathbf{1}_{n \times n}$, *i.e.*, every element is 1. Thus, we can calculate its sample variance via the 1-norm of the message matrix.

We consider that the message-passing GNNs do not possess the node-sampler or the edge-sampler, which means every directed edge corresponds to a row vector in the message matrix $\mathbf{M}$. For analytical simplicity, we assume that the graph is undirected and the degree of each node is d. In this case, $k = 2|E| = nd$ rows of the message matrix counts in total. All random dropping methods can be considered as multiple independent Bernoulli samplings. The whole process conforms to a binomial distribution, and so we can calculate the variance of $|\mathbf{M}|$.

Dropout. Perform nc times of Bernoulli sampling. Dropping an element in the feature matrix leads to masking d elements in the message matrix. Its variance can be calculated by $Var_{do}(|\mathbf{M}|) = (1 - \delta)\delta ncd^2$.

DropEdge. Perform $\frac{nd}{2}$ times of Bernoulli sampling. Dropping an element in the adjacency matrix leads to masking $2c$ elements in the message matrix. Its variance can be calculated by $Var_{de}(|\mathbf{M}|) = 2(1 - \delta)\delta nc^2 d$.

DropNode. Perform n times of Bernoulli sampling. Dropping an element in the node set leads to masking cd elements in the message matrix. Its variance can be calculated by $Var_{dn}(|\mathbf{M}|) = (1 - \delta)\delta nc^2 d^2$.

DropMessage. Perform ncd times of Bernoulli sampling. Dropping an element in the message matrix leads to masking 1 elements in the message matrix. Its variance can be calculated by $Var_{dm}(|\mathbf{M}|) = (1-\delta)\delta ncd$.

Therefore, the variances of the random dropping methods are sorted as follows:

$$Var_{dm}(|\mathbf{M}|) \leq Var_{do}(|\mathbf{M}|) \leq Var_{dn}(|\mathbf{M}|)$$

$$Var_{dm}(|\mathbf{M}|) \leq Var_{de}(|\mathbf{M}|)$$

Our DropMessage has the smallest sample variance among all existing random dropping methods. □

8.2 Experiment Details

Hardware spcification and environment. We run our experiments on the machine with Intel Xeon Gold CPUs (6240@2.6Ghz), ten NVIDIA GeForce 2080ti GPUs (11GB). The code is written in Python 3.8, Pytorch 1.8, and Pytorch-Geometric 1.7.

Dataset statistics. Table 4 shows the statistics of datasets.

Table 4: Dataset Statistics.

Dataset	Nodes	Edges	Feature	Classes	Train/Val/Test
Cora	2708	5429	1433	7	140 / 500 / 1000
CiteSeer	3327	4732	3703	6	120 / 500 / 1000
PubMed	19717	44338	500	3	60 / 500 / 1000
ogbn-arxiv	169343	1166243	128	40	90941 / 29799 / 48603
Flickr	89250	899756	500	7	50% / 25% / 25%
FinV	340751	1575498	261	2	60% / 20% / 20%
Telecom	509304	809996	21	2	60% / 20% / 20%

Implementation details. We conduct 20 independent experiments for each setting and obtain the average results. On the five public datasets, we continue to employ the same hyper-parameter settings as previous works have proposed. And on the two real-world datasets, we obtain the best parameters through careful tuning. As for public datasets *Cora*, *CiteSeer*, *PubMed*, and *Flickr*, we apply two-layer models. However, when it comes to the public dataset *ogbn-arxiv* and two industrial datasets, *Telecom* and *FinV*, we employ three-layer models with two batch normalization layers between the network layers. These experimental settings are identical for node classification tasks and link prediction tasks. The number of hidden units on GCNs is 16 for *Cora*, *CiteSeer*, *PubMed*, and is 64 for others. For GATs, we apply eight-head models with 8 hidden units for *Cora*, *CiteSeer*, *PubMed*, and use single-head models with 128 hidden units for the other datasets. As for APPNPs, we use the teleport probability $\alpha = 0.1$ and $K = 10$ power iteration steps. The number of hidden units on APPNPs is always 64 for all datasets. In all cases, we use Adam optimizers with learning rate of 0.005 and L_2 regularization of 5×10^{-4}, and train each model 200 epochs. We adjust the dropping rate from 0.05 to 0.95 in steps of 0.05 and select the optimal one for each setting. Table 5 and 6 present the optimal selections of dropping rates δ.

Table 5: Statistics of Optimal Dropping Rates δ for Link Predictions.

Dataset	Dropout	DropEdge	DropNode	DropMessage
GCN-Cora	0.20	0.45	0.40	0.40
GAT-Cora	0.20	0.40	0.15	0.45
APPNP-Cora	0.50	0.35	0.05	0.35
GCN-CiteSeer	0.25	0.30	0.25	0.50
GAT-CiteSeer	0.25	0.50	0.15	0.50
APPNP-CiteSeer	0.50	0.50	0.10	0.40
GCN-PubMed	0.25	0.25	0.50	0.25
GAT-PubMed	0.50	0.15	0.25	0.50
APPNP-PubMed	0.20	0.25	0.05	0.20

Additional data for comparison results. For the node classification task, the performance is measured by accuracy on four public datasets (Cora, CiteSeer, PubMed, ogbn-arxiv). As for Flickr and two imbalanced industrial datasets, we employ F1 scores. When it comes to the link prediction task, we calculate the AUC values for comparisons. Table 7 present the std values of comparison results.

Table 6: Statistics of Optimal Dropping Rates δ for Node Classifications.

Dataset	Dropout	DropEdge	DropNode	DropMessage
GCN-Cora	0.90	0.40	0.90	0.90
GAT-Cora	0.90	0.40	0.25	0.90
APPNP-Cora	0.85	0.70	0.15	0.80
GCN-CiteSeer	0.80	0.40	0.85	0.90
GAT-CiteSeer	0.35	0.90	0.45	0.90
APPNP-CiteSeer	0.85	0.45	0.35	0.80
GCN-PubMed	0.20	0.10	0.10	0.15
GAT-PubMed	0.60	0.70	0.35	0.85
APPNP-PubMed	0.80	0.50	0.10	0.75
GCN-ogbn-arxiv	0.20	0.20	0.15	0.25
GAT-ogbn-arxiv	0.20	0.20	0.15	0.20
APPNP-ogbn-arxiv	0.20	0.20	0.20	0.25
GCN-Flickr	0.15	0.20	0.15	0.20
GAT-Flickr	0.35	0.15	0.05	0.40
APPNP-Flickr	0.35	0.35	0.05	0.35

Results of robustness analysis. We conduct experiments for robustness analysis on three citation networks: Cora, CiteSeer, and PubMed. Specifically, we randomly add a certain ratio of edges (0%, 10%, 20%, 30%) into these datasets and perform the node classification. Table 8 summarizes the classification accuracy of robustness analysis.

8.3 Related Works About Data Augmentations

In this section, we introduce some previous works about data augmentation techniques that are related to the random dropping methods discussed in our paper. GNN's effectiveness tends to be weakened due to the noise and low-resource problems in real-world graph data (Dai, Aggarwal, and Wang 2021)(Ding et al. 2022a)(Sun, Lin, and Zhu 2020). Data augmentation has attracted a lot of research interest as it is an effective tool to improve model performance in noisy settings (Zhao et al. 2021). However, apart from i.i.d. data, graph data, which is defined on non-Euclidean space with multi-modality, is hard to be handled by conventional data augmentation methods (Ding et al. 2022c). To address this problem, an increasing number of graph data augmentation methods have been proposed, which include feature-wise (Velickovic et al. 2019a), structure-wise (Cai, Wang, and Wang 2021)(Jin et al. 2021), and label-wise augmentations (Zhang et al. 2017)(Verma et al. 2019).

8.4 Additional Experiments

Results on SOTA models. We also compare the performance of different random dropping methods and our proposed DropMessage on two SOTA backbone models: DAGNN (Liu, Gao, and Ji 2020) and GCNII (Ming Chen, Zengfeng Huang, and Li 2020). We employ these two models to perform node classification tasks on five datasets.

Table 9 presents the experimental results. The results indicate that DropMessage consistently outperforms other random dropping methods.

Comparison to Random Augmentation Methods. Random dropping methods are similar to random augmentation techniques used in graph contrastive learning. We compare the performance of our proposed DropMessage with some widely-used augmentation techniques (Ding et al. 2022b), and their brief descriptions are listed as below.

Node Dropping: it randomly discards a certain portion of vertices along with their connections.

Edge Perturbation: it perturbs the connectivities in graph through randomly adding or dropping a certain ratio of edges.

Subgraph: it samples a subgraph using random walk.

Table 10 summaries the experimental results on five datasets (Cora, CiteSeer, PubMed, ogbn-arxiv and Flickr) for node classification tasks. The results indicate that our proposed DropMessage consistently outperforms the random data augmentation methods.

Graph Property Prediction. We perform the graph property prediction task on ogbg-molhiv and ogbg-molpcba (Hu et al. 2020). They are two molecular property prediction datasets adopted from the MoleculeNet (Wu et al. 2017). Table 11 shows the results of ROC-AUC scores with GCN as the backbone model.

Graph Rewiring. We conduct experiments on graph rewiring to evaluate the robustness of random dropping methods. Specifically, we first remove a certain ratio of edges, and then randomly add an equal number of edges. We perform the experiments on three citation datasets with GCN as the backbone model. Table 12 presents the results.

Table 7: The Std Values of Comparison Results.

Task & Dataset / Model	Node classification							Link prediction		
	Cora	CiteSeer	PubMed	ogbn-arxiv	Flickr	Telecom	FinV	Cora	CiteSeer	PubMed
GCN	0.37	0.55	0.67	0.41	0.0044	0.0043	0.0061	0.0022	0.0053	0.0043
GCN-Dropout	0.68	0.59	0.77	0.52	0.0050	0.0067	0.0069	0.0044	0.0073	0.0055
GCN-DropEdge	0.91	0.72	0.81	0.54	0.0061	0.0055	0.0077	0.0045	0.0077	0.0080
GCN-DropNode	1.06	0.95	0.88	0.75	0.0064	0.0079	0.0089	0.0064	0.0091	0.0094
GCN-DropMessage	0.59	0.59	0.53	0.56	0.0042	0.0027	0.0055	0.0044	0.0064	0.0051
GAT	0.59	0.62	0.53	0.37	0.0042	0.0033	0.0043	0.0042	0.0047	0.0049
GAT-Dropout	0.77	0.71	0.67	0.49	0.0063	0.0046	0.0067	0.0054	0.0062	0.0066
GAT-DropEdge	0.82	1.02	0.79	0.54	0.0077	0.0052	0.0055	0.0079	0.0082	0.0087
GAT-DropNode	0.89	0.86	0.88	0.62	0.0079	0.0077	0.0091	0.0093	0.0092	0.0085
GAT-DropMessage	0.69	0.67	0.47	0.54	0.0052	0.0055	0.0056	0.0037	0.0032	0.0077
APPNP	0.33	0.34	0.51	0.33	0.0024	0.0032	0.0053	0.0038	0.0017	0.0069
APPNP-Dropout	0.43	0.54	0.43	0.29	0.0032	0.0059	0.0072	0.0063	0.0046	0.0042
APPNP-DropEdge	0.72	0.71	0.86	0.55	0.0039	0.0032	0.0088	0.0087	0.0058	0.0098
APPNP-DropNode	0.49	0.42	0.66	0.73	0.0054	0.0065	0.0077	0.0074	0.0035	0.0103
APPNP-DropMessage	0.52	0.24	0.37	0.38	0.0044	0.0039	0.0044	0.0065	0.0041	0.0032

Table 8: Classification Accuracy (%) for Robustness Analysis.

Dataset / Model	Cora				CiteSeer				PubMed			
	0%	10%	20%	30%	0%	10%	20%	30%	0%	10%	20%	30%
GCN	80.68	78.51	76.72	75.36	70.83	68.66	66.32	65.15	78.97	75.55	73.18	72.11
GCN-Dropout	_83.16_	_80.97_	_78.17_	_76.83_	71.48	69.86	67.35	66.08	_79.13_	_76.94_	_74.94_	_74.07_
GCN-DropEdge	81.69	79.45	77.47	76.44	71.43	69.60	67.26	66.14	79.06	76.57	74.88	73.93
GCN-DropNode	83.04	80.13	78.12	76.72	**72.12**	**70.51**	**68.21**	**66.94**	79.00	76.74	74.71	73.86
GCN-DropMessage	**83.33**	**81.04**	**79.09**	**77.26**	_71.83_	_70.08_	_67.61_	_66.49_	**79.20**	**77.10**	**75.02**	**74.11**
GAT	81.35	78.14	76.48	74.56	70.14	67.51	64.99	63.65	77.20	75.05	72.81	71.59
GAT-Dropout	**82.41**	**80.20**	**78.71**	**77.23**	71.31	68.38	66.84	64.92	**78.31**	_76.05_	_74.14_	_72.88_
GAT-DropEdge	81.82	79.08	76.92	75.32	71.17	_69.07_	_67.21_	_65.31_	77.70	75.92	74.02	72.73
GAT-DropNode	82.08	78.80	76.98	75.84	_71.44_	68.57	66.42	64.68	77.98	75.87	73.57	72.38
GAT-DropMessage	_82.20_	_79.70_	_78.11_	_76.53_	**71.48**	**69.24**	**67.47**	**65.49**	_78.14_	**76.20**	**74.22**	**72.97**
APPNP	81.45	77.75	75.61	73.54	70.62	65.76	62.60	60.92	79.79	75.29	72.77	71.06
APPNP-Dropout	82.23	79.06	76.55	74.30	71.93	66.55	63.22	61.61	_79.92_	76.45	74.12	72.17
APPNP-DropEdge	**82.75**	_78.90_	**76.63**	**74.68**	_72.10_	_66.58_	**63.27**	**61.77**	79.83	_76.72_	_74.17_	_72.21_
APPNP-DropNode	81.79	78.17	75.79	73.76	71.50	65.86	63.01	61.01	79.81	76.45	74.05	72.01
APPNP-DropMessage	_82.37_	**79.12**	_76.60_	_74.59_	**72.65**	**66.74**	_63.25_	_61.59_	**80.04**	**76.73**	**74.25**	**72.25**

Table 9: Results on SOTA Models.

Dataset / Model	Cora	CiteSeer	PubMed	ogbn-arxiv	Flickr
DAGNN	82.73 ± 0.27	72.82 ± 0.43	80.37 ± 0.32	71.60 ± 0.19	0.5243 ± 0.0017
DAGNN-Dropout	84.50 ± 0.50)	73.30 ± 0.58	80.56 ± 0.52	72.09 ± 0.25	0.5324 ± 0.0031
DAGNN-DropEdge	83.78 ± 0.55	73.02 ± 0.50	80.27 ± 0.42	71.88 ± 0.21	0.5311 ± 0.0029
DAGNN-DropNode	83.97 ± 0.45	73.41 ± 0.67	80.42 ± 0.40	72.03 ± 0.27	0.5304 ± 0.0016
DAGNN-DropMessage	**84.64 ± 0.61**	**73.35 ± 0.69**	**80.58 ± 0.58**	**72.23 ± 0.33**	**0.5325 ± 0.0024**
GCNII	82.24 ± 0.15	72.11 ± 0.41	79.85 ± 0.20	72.37 ± 0.13	0.5143 ± 0.0032
GCNII-Dropout	85.45 ± 0.50	73.42 ± 0.55	80.18 ± 0.37	72.74 ± 0.16	0.5170 ± 0.0029
GCNII-DropEdge	84.96 ± 0.46	72.98 ± 0.42	80.05 ± 0.29	72.41 ± 0.15	0.5180 ± 0.0023
GCNII-DropNode	85.15 ± 0.66	73.44 ± 0.48	79.87 ± 0.21	72.38 ± 0.21	0.5159 ± 0.0029
GCNII-DropMessage	**85.53 ± 0.62**	**73.28 ± 0.51**	**80.21 ± 0.33**	**72.78 ± 0.16**	**0.5192 ± 0.0026**

Table 10: Results of Random Augmentation Methods.

Model \ Dataset	Cora	CiteSeer	PubMed	ogbn-arxiv	Flickr
GCN	80.68 ± 0.37	70.83 ± 0.55	78.97 ± 0.67	70.08 ± 0.41	0.5188 ± 0.0044
GCN-DropMessage	$\mathbf{83.33 \pm 0.59}$	$\mathbf{71.83 \pm 0.59}$	$\mathbf{79.20 \pm 0.53}$	$\mathbf{71.27 \pm 0.56}$	$\mathbf{0.5223 \pm 0.0042}$
GCN-NodeDropping	81.06 ± 0.53	71.20 ± 0.33	79.06 ± 0.25	70.91 ± 0.54	0.5210 ± 0.0065
GCN-EdgePerturbation	82.12 ± 0.54	71.70 ± 0.41	79.08 ± 0.74	70.69 ± 0.62	0.5205 ± 0.0051
GCN-Subgraph	77.40 ± 0.47	68.88 ± 0.46	77.44 ± 0.58	69.34 ± 0.69	0.5087 ± 0.0061
GAT	81.35 ± 0.59	70.14 ± 0.62	77.20 ± 0.53	70.32 ± 0.37	0.4988 ± 0.0042
GAT-DropMessage	$\mathbf{82.20 \pm 0.69}$	$\mathbf{71.48 \pm 0.67}$	$\mathbf{78.14 \pm 0.47}$	$\mathbf{71.13 \pm 0.54}$	$\mathbf{0.5013 \pm 0.0052}$
GAT-NodeDropping	80.68 ± 0.58	70.42 ± 1.77	78.08 ± 0.22	70.80 ± 0.67	0.4990 ± 0.0051
GAT-EdgePerturbation	81.92 ± 0.60	70.52 ± 1.02	78.04 ± 0.54	70.44 ± 0.47	0.5001 ± 0.0046
GAT-Subgraph	77.48 ± 0.41	69.20 ± 0.34	78.04 ± 0.21	68.56 ± 0.88	0.4914 ± 0.0060
APPNP	81.45 ± 0.33	70.62 ± 0.34	79.79 ± 0.51	69.11 ± 0.33	0.5047 ± 0.0024
APPNP-DropMessage	$\mathbf{82.37 \pm 0.52}$	$\mathbf{72.65 \pm 0.24}$	$\mathbf{80.04 \pm 0.37}$	$\mathbf{69.72 \pm 0.38}$	$\mathbf{0.5072 \pm 0.0044}$
APPNP-NodeDropping	81.94 ± 0.91	71.36 ± 0.28	79.18 ± 0.27	69.23 ± 0.54	0.5041 ± 0.0058
APPNP-EdgePerturbation	80.10 ± 0.61	72.12 ± 0.23	78.90 ± 0.18	69.04 ± 0.35	0.5052 ± 0.0043
APPNP-Subgraph	80.28 ± 0.58	69.80 ± 0.51	76.64 ± 1.57	67.23 ± 1.06	0.4833 ± 0.0089

Table 11: ROC-AUC Scores for Graph Property Prediction.

Model \ Dataset	ogbg-molhiv	ogbg-molpcba
GCN	0.7581 ± 0.0061	0.2013 ± 0.0012
GCN-Dropout	0.7606 ± 0.0097	0.2020 ± 0.0024
GCN-DropEdge	0.7602 ± 0.0093	0.2015 ± 0.0017
GCN-DropNode	0.7592 ± 0.0087	0.2019 ± 0.0022
GCN-DropMessage	$\mathbf{0.7614 \pm 0.0103}$	$\mathbf{0.2027 \pm 0.0041}$

Table 12: Classification Accuracy (%) for Rewiring Graphs.

Model \ Dataset	Cora				CiteSeer				PubMed			
	0%	10%	20%	30%	0%	10%	20%	30%	0%	10%	20%	30%
GCN	80.68	77.36	74.64	71.96	70.83	68.46	65.84	62.94	78.97	76.14	72.70	71.70
GCN-Dropout	83.16	79.06	75.90	72.84	71.48	69.14	66.24	63.20	79.13	76.60	73.92	72.46
GCN-DropEdge	81.69	78.04	75.02	72.10	71.43	69.30	66.18	63.18	79.06	76.26	73.90	72.36
GCN-DropNode	83.04	78.16	75.82	72.74	**72.12**	**69.84**	**66.96**	**63.48**	79.00	76.64	74.06	72.48
GCN-DropMessage	**83.33**	**79.34**	**76.24**	**73.28**	71.83	69.52	66.40	63.38	**79.20**	**76.74**	**74.24**	**72.60**

Adversarial Attacks. We also conduct experiments to evaluate the effectiveness of random dropping methods against adversarial attacks. We apply PGD attacks (Xu et al. 2019) to perturb the graph structures on Cora and CiteSeer, using GCN as the backbone model. Figure 3 presents the results.

8.5 Theoretical Analysis Towards Over-smoothing

The over-smoothing issue extensively exists in graph neural networks (Chen et al. 2020; Elinas and Bonilla 2022). As the number of model layers increases, node representations become nearly indistinguishable, which leads to a significant decrease on model performance. However, random dropping methods can alleviate this problem, and our proposed DropMessage achieves the best effect compared with other random dropping methods.

Now, we give a theoretical demonstration from the perspective of the information theory by measuring the Shannon entropy (Shannon 2001) of propagated messages. The Shannon entropy quantifies "the amount of information", and measures the degree of confusion and diversity of the given variable, which can be expressed as:
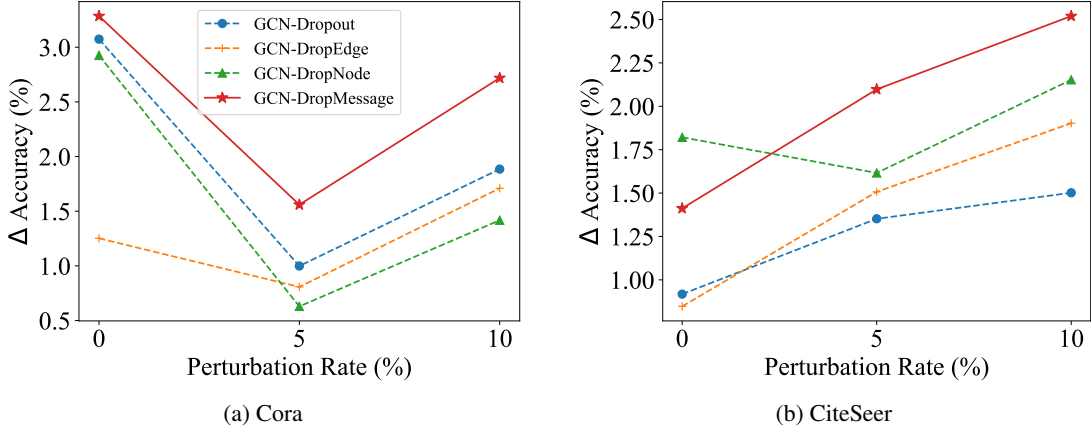
Figure 3: Model Performances Against PGD Attacks.

$$H(X) := \mathbb{E}[-\log p(X)] = -\sum_{x \in \mathcal{X}} p(x) \log p(x)$$

where $\mathcal{X}$ denotes all possible values of x. Now, we measure the degree of over-smoothing of the model by calculating the Shannon entropy of the propagated messages. Intuitively, the larger the Shannon entropy, the more diverse the propagated messages are, and the less likely the aggregated representations will converge.

We assume there are k types of d-dimension messages propagated in the GNN model. For the i-th type of the propagated messages, they are delivered for t_i times by n_i nodes. Then, we can calculate the Shannon entropy for the initial messages:

$$H(clean) = \sum_i^k -p_i log\, p_i$$

Then, we consider the Shannon entropy after random dropping with dropping rate δ. For DropEdge and DropNode, they generate blank messages at the ratio δ. So the Shannon entropy can be expressed as:

$$E(H(DropEdge)), E(H(DropNode)) = -\delta log(\delta) + (1-\delta)\sum_i^k -p_i log((1-\delta)p_i)$$

When it comes to Dropout, the Shannon entropy can be calculated as:

$$E(H(Dropout)) = \delta \sum_i^k -p_i log(p_i/min\{d, n_i\}) + (1-\delta)\sum_i^k -p_i log((1-\delta)p_i)$$

As for our proposed DropMessage, the Shannon entropy can be expressed as:

$$E(H(DropMessage)) = \delta \sum_i^k -p_i log(p_i/min\{d, t_i\}) + (1-\delta)\sum_i^k -p_i log((1-\delta)p_i)$$

Besides, we have $t_i \geq n_i$. So, with proper dropping rate δ, we can obtain:

$$E(H(DropMessage)) \geq E(H(Dropout)), E(H(DropEdge)), E(H(DropNode)) \geq H(clean)$$

From above derivations, we prove the effectiveness of random dropping methods in alleviating over-smoothing issue, and our proposed DropMessage achieves the best effect.