Graph Random Neural Networks

Wenzheng Feng[1*], Jie Zhang[2*‡], Yuxiao Dong[3], Yu Han[1], Huanbo Luan[1], Qian Xu[2], Qiang Yang[2], Evgeny Kharlamov[4], Jie Tang[1§]

[1] Department of Computer Science and Technology, Tsinghua University

[2]WeBank Co., Ltd [3]Microsoft Research [4]Bosch Center for Artificial Intelligence

fwz17@mails.tsinghua.edu.cn, {zhangjie.exe, ericdongyx, luanhuanbo}@gmail.com

yuhanthu@126.com, {qianxu, qiangyang}@webank.com, evgeny.kharlamov@de.bosch.com, jietang@tsinghua.edu.cn

ABSTRACT

Graph neural networks (GNNs) have generalized deep learning methods into graph-structured data with promising performance on graph mining tasks. However, existing GNNs often meet complex graph structures with scarce labeled nodes and suffer from the limitations of non-robustness [51, 53], over-smoothing [8, 29, 30], and overfitting [19, 30]. To address these issues, we propose a simple yet effective GNN framework—Graph Random Neural Network (Grand). Different from the deterministic propagation in existing GNNs, Grand adopts a random propagation strategy to enhance model robustness. This strategy also naturally enables Grand to decouple the propagation from feature transformation, reducing the risks (减弱) of over-smoothing and overfitting. Moreover, random propagation acts as an efficient method for graph data augmentation. Based on this, we propose the consistency regularization for Grand by leveraging the distributional consistency of unlabeled nodes in multiple augmentations, improving the generalization capacity of the model. Extensive experiments on graph benchmark datasets suggest that Grand significantly outperforms state-of-the-art GNN baselines on semi-supervised graph learning tasks. Finally, we show that Grand mitigates the issues of over-smoothing and overfitting, and its performance is married with robustness.

1 INTRODUCTION

The success of deep learning has provoked interest to generalize neural networks to structured data, marking the emergence of graph neural networks (GNNs) [6, 17, 20]. Over the course of its development, GNNs have been shifting the paradigm of graph mining from structural explorations to representation learning. A wide variety of graph applications have benefited from this shift, such as node classification [10, 27], link prediction [45, 48, 50], and graph classification [32, 46].

The essential procedure in GNNs is the feature propagation, which is usually performed by some deterministic propagation rules derived from the graph structures. For example, the graph convolutional network (GCN) [27] propagates information based on the normalized Laplacian of the input graph, which is coupled with the feature transformation process. Such a propagation can be also viewed as modeling each node's neighborhood as a receptive field and enabling a recursive neighborhood propagation process by stacking multiple GCN layers. Further, this process can be also unified into the neural message passing framework [17].

Recent attempts to advance the propagation based architecture include adding self-attention (GAT) [42], integrating with graphical models (GMNN) [36], and neighborhood mixing (MixHop) [2], etc. However, while the propagation procedure can enable GNNs to achieve attractive performance, it also brings some inherent issues that have been recognized recently, including non-robustness [51, 53], over-smoothing [8, 29, 30], and overfitting [19, 30].

Non-robustness. The deterministic propagation in most GNNs naturally makes each node to be highly dependent with its (multi-hop) neighborhoods. This leaves the nodes to be easily misguided by potential data noise, making GNNs non-robust [51]. For example, it has been shown that GNNs are very susceptible to adversarial attacks, and the attacker can indirectly attack the target node by manipulating long-distance neighbors [53].

Overfitting. To make the node representations more expressive, it is often desirable to increase GNNs' layers so that information can be captured from high-order neighbors. However, in each GNN layer, the propagation process is coupled with the non-linear transformation. Therefore, stacking many layers for GNNs can bring more parameters to learn and thus easily cause overfitting [19, 30].

Over-smoothing. In addition, recent studies show that the convolution operation is essentially a special form of Laplacian smoothing [29], which propagates neighbors' features into the central node. As a result, directly stacking many layers tend to make nodes' features over-smoothed [8, 30]. In other words, each node incorporates too much information from others but loses the specific information of itself, making them indistinguishable after the propagation.

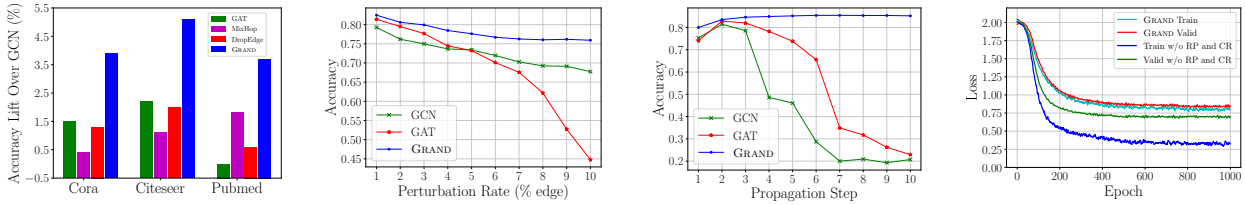
These challenges are further amplified under the semi-supervised setting, wherein the supervision information is scarce [7]. Though several efforts have been devoted, such as DropEdge [37], these problems remain largely unexplored and, in particular, unresolved in a systemic way. In this work, we propose to systemically address all these fundamental issues for graph neural networks. We achieve this by questioning and re-designing GNNs' core procedure—the (deterministic) graph propagation. Specifically, we present the Graph Random Neural Networks (Grand) for semi-supervised learning on graphs. Grand comprises two major components: random propagation (RP) and consistency regularization (CR).

First, we introduce a simple yet effective message passing strategy—random propagation—which allows each node to randomly drop the entire features of some (multi-hop) neighbors during each training epoch. As such, each node is enabled to be not sensitive to specific neighborhoods, *increasing the robustness of Grand*. Second, the design of random propagation can naturally

*Equal contribution.

‡Work performed while at Tsinghua University.

§Corresponding author.



(a) Performance improvements over GCN (b) Robustness under random attack (c) Mitigation of over-smoothing (d) Generalization improvement

Figure 1: GRAND's performance, robustness, and mitigation of over-smoothing & overfitting demonstrated.

separate feature propagation and transformation, which are commonly coupled with each other in most GNNs. This empowers GRAND to safely perform higher-order feature propagation without increasing the complexity, *reducing the risk of over-smoothing and overfitting*. Finally, we demonstrate that random propagation is an economic data augmentation method on graphs, based on which we propose the *consistency regularized training* for GRAND. This strategy enforces the model to output similar predictions on different data augmentations of the same data, *further improving GRAND's generalization capability under the semi-supervised setting.*

To demonstrate the performance of GRAND, we conduct extensive experiments for semi-supervised graph learning on three GNN benchmark datasets, as well as six publicly available large datasets (Cf. Appendix A.3). In addition, we also provide theoretical analyses to understand the effects of the proposed random propagation and consistency regularization strategies on GNNs.

Figure 1 illustrates the GRAND's advantages in terms of performance, robustness, and mitigation of over-smoothing and overfitting. (a) **Performance**: GRAND achieves a 3.9% improvement (absolute accuracy gap) over GCN on Cora, while the margins lifted by GAT and DropEdge were only 1.5% and 1.3%, respectively. (b) **Robustness**: As more random edges injected into the data, GRAND experiences a weak accuracy-declining trend, while the decline of GCN and GAT is quite sharp. (c) **Over-smoothing**: As more layers stacked, the accuracies of GCN and GAT decrease dramatically—from 0.75 to 0.2—due to over-smoothing, while GRAND actually benefits from more propagation steps. (d) **Overfitting**: Empowered by the random propagation and consistency regularization, GRAND's training and validation cross-entropy losses converge close to each other, while a significant gap between the losses can be observed when RP and CR are removed.

In summary, GRAND outperforms state-of-the-art GNNs in terms of effectiveness and robustness, and mitigates the over-smoothing and overfitting issues that are commonly faced in existing GNNs.

2 PROBLEM AND RELATED WORK

Let $G = (V, E)$ denote a graph, where V is a set of $|V| = n$ nodes and $E \subseteq V \times V$ is a set of $|E|$ edges between nodes. $\mathbf{A} \in \{0,1\}^{n\times n}$ denotes G's adjacency matrix, with each element $\mathbf{A}_{ij} = 1$ indicating there exists an edge between v_i and v_j, otherwise $\mathbf{A}_{ij} = 0$.

This work focuses on semi-supervised graph learning, in which each node v_i is associated with 1) a feature vector $\mathbf{X}_i \in \mathbf{X} \in \mathbb{R}^{n\times d}$ and 2) a label vector $\mathbf{Y}_i \in \mathbf{Y} \in \{0,1\}^{n\times C}$ with C representing the number of classes. For semi-supervised classification, m nodes ($0 < m \ll n$) have observed their labels $\mathbf{Y}^L$ and the labels $\mathbf{Y}^U$ of the remaining $n - m$ nodes are missing.

Semi-Supervised Node Classification. Given a partially labeled graph $G = (V, E)$ with the node feature matrix $\mathbf{X}$, and observed labels $\mathbf{Y}^L$, the objective is to learn a predictive function $f : G, \mathbf{X}, \mathbf{Y}^L \rightarrow \mathbf{Y}^U$ to infer the missing labels $\mathbf{Y}^U$ for unlabeled nodes.

推断

Recently, a significant line of related work has been devoted into this task, which is reviewed in the following section.

Graph Neural Networks. Graph neural networks (GNNs) [20, 27, 38] generalize neural techniques into graph-structured data. The core operation in GNNs is graph propagation, in which information is propagated from each node to its neighborhoods with some deterministic propagation rules. For example, the graph convolutional network (GCN) [27] adopts the following propagation rule:

$$\mathbf{H}^{(l+1)} = \sigma(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}), \tag{1}$$

where $\hat{\mathbf{A}}$ is the symmetric normalized adjacency matrix, $\mathbf{W}^{(l)}$ is the weight matrix of the l^{th} layer, and $\sigma(.)$ denotes ReLU function. $\mathbf{H}^{(l)}$ is the hidden node representation in the l^{th} layer with $\mathbf{H}^{(0)} = \mathbf{X}$. The propagation in Eq. 1 could be explained via 1) an approximation of the spectral graph convolutional operations [6, 12, 23], 2) neural message passing [17], and 3) convolutions on direct neighborhoods [22, 34]. Recent attempts to advance this architecture include GAT [42], GMNN [36], MixHop [2], GraphNAS [16], and so on. Often, these models face the challenges of overfitting and over-smoothing due to the deterministic graph propagation process [8, 29, 30]. Differently, we propose random propagation for GNNs, which decouples feature propagation and non-linear transformation in Eq. 1, reducing the risk of over-smoothing and overfitting. Recent efforts have also been devoted to performing node sampling for fast and scalable GNN training, such as GraphSAGE [22], FastGCN [9], AS-GCN [24], and LADIES [52]. Different from these work, in this paper, a new sampling strategy DropNode, is proposed for improving the robustness and generalization of GNNs for semi-supervised learning. Compared with GraphSAGE's node-wise sampling, DropNode 1) enables the decoupling of feature propagation and transformation, and 2) is more efficient as it does not require recursive sampling of neighborhoods for every node. Finally, it drops each node based an i.i.d. Bernoulli distribution, differing from the importance sampling in FastGCN, AS-GCN, and LADIES.

Regularization Methods for GCNs. Broadly, a popular regularization method in deep learning is data augmentation, which expands the training samples by applying some transformations or

injecting noise into input data [14, 19, 44]. Based on data augmentation, we can further leverage consistency regularization [3, 28] for semi-supervised learning, which enforces the model to output the same distribution on different augmentations of an example. Following this idea, a line of work has aimed to design powerful regularization methods for GNNs, such as VBAT [13], G^{3}NN [31], GraphMix [43], and DropEdge [37]. For example, GraphMix [43] introduces the MixUp [49] for training GCNs. Different from GRAND, GraphMix augments graph data by performing linear interpolation between two samples in hidden space, and regularizes the GCNs by encouraging the model to predict the same interpolation of corresponding labels. DropEdge [37] aims to militate over-smoothing by randomly removing some edges during training but does not bring significant performance gains for semi-supervised learning task. However, DropNode is designed to 1) enable the separation of feature propagation and transformation for random feature propagation and 2) further augment graph data augmentations and facilitate the consistency regularized training.

3 GRAPH RANDOM NETWORKS

In this section, we present the GRAPH RANDOM NEURAL NETWORKS (GRAND) for semi-supervised learning on graphs. Its idea is to enable each node to randomly propagate with different subsets of neighbors in different training epochs. This random propagation strategy is demonstrated as an economic way for stochastic graph data augmentation, based on which we design a consistency regularized training for improving GRAND's generalization capacity.

Figure 2 illustrates the full architecture of GRAND. Given an input graph, GRAND generates multiple data augmentations by performing random propagation (DropNode + propagation) multiple times at each epoch. In addition to the classification loss, GRAND also leverages a consistency regularization loss to enforce the models to give similar predictions across different augmentations.

3.1 Random Propagation

Given an input graph with its associated feature matrix, GRAND first conducts the random propagation process and then makes the prediction by using the simple multilayer perceptron (MLP) model.

The motivation for random propagation is to address the non-robustness issue faced by existing GNNs [11, 53, 54]. This process is coupled with the DropNode and propagation steps. In doing so, GRAND naturally separates the feature propagation and non-linear transformation operations in standard GNNs, enabling GRAND to reduce the risk of the overfitting and over-smoothing issues. ? ? ?

DropNode. In random propagation, we aim to perform message passing in a random way during model training such that each node is not sensitive to specific neighborhoods. To achieve this, we design a simple yet effective node sampling operation—DropNode—before the propagation layer.

DropNode is designed to randomly remove some nodes' all features. In specific, at each training epoch, the entire feature vector of each node is randomly discarded with a pre-defined probability, i.e., some rows of $\mathbf{X}$ are set to $\vec{0}$. The resultant perturbed feature matrix $\widetilde{\mathbf{X}}$ is then fed into the propagation layer.

The formal DropNode operation is shown in Algorithm 1. First, we randomly sample a binary mask $\epsilon_i \sim Bernoulli(1-\delta)$ for each node v_i. Second, we obtain the perturbed feature matrix $\widetilde{\mathbf{X}}$ by multiplying each node's feature vector with its corresponding mask, i.e., $\widetilde{\mathbf{X}}_i = \epsilon_i \cdot \mathbf{X}_i$. Finally, we scale $\widetilde{\mathbf{X}}$ with the factor of $\frac{1}{1-\delta}$ to guarantee the perturbed feature matrix is in expectation equal to $\mathbf{X}$. Note that the sampling procedure is only performed during training. During inference, we directly set $\widetilde{\mathbf{X}}$ with the original feature matrix $\mathbf{X}$.

伯努力分布

$$f_X(x) = p^x(1-p)^{1-x} = \begin{cases} p & \text{if } x = 1, \\ q & \text{if } x = 0. \end{cases}$$

Algorithm 1 DropNode

Input:
Feature matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, DropNode probability $\delta \in (0, 1)$.

Output:
Perturbed feature matrix $\widetilde{\mathbf{X}} \in \mathbb{R}^{n \times d}$.

1: Randomly sample n masks: $\{\epsilon_i \sim Bernoulli(1-\delta)\}_{i=0}^{n-1}$.
2: Obtain deformity feature matrix by multiplying each node's feature vector with the corresponding mask: $\widetilde{\mathbf{X}}_i = \epsilon_i \cdot \mathbf{X}_i$.
3: Scale the deformity features: $\widetilde{\mathbf{X}} = \frac{\widetilde{\mathbf{X}}}{1-\delta}$.

where $\hat{A}$ is the symmetric normalized adjacency matrix
对称归一化邻接矩阵

After DropNode, the perturbed feature matrix $\widetilde{\mathbf{X}}$ is fed into the propagation layer to perform message passing. Here we adopt mixed-order propagation, i.e., $\overline{\mathbf{X}} = \overline{\mathbf{A}}\widetilde{\mathbf{X}}$, where $\overline{\mathbf{A}} = \sum_{k=0}^{K} \frac{1}{K+1}\hat{\mathbf{A}}^k$—the average of the power series of $\hat{\mathbf{A}}$ from order 0 to order K. This kind of propagation rule enables the model to incorporate the multi-order neighborhood information, reducing the risk of over-smoothing when compared with using $\hat{\mathbf{A}}^K$ only. Similar ideas have been adopted in recent GNN studies [1, 2].

Prediction Module. After the random propagation module, the augmented feature matrix $\overline{\mathbf{X}}$ can be then fed into any neural networks for predicting nodes labels. In GRAND, we employ a two-layer MLP as the classifier, that is:

$$P(\mathbf{Y}|\overline{\mathbf{X}};\Theta) = \sigma_2(\sigma_1(\overline{\mathbf{X}}\mathbf{W}^{(1)})\mathbf{W}^{(2)}) \tag{2}$$

where $\sigma_1(.)$ is the ReLU function, $\sigma_2(.)$ is the softmax function, and $\Theta = \{\mathbf{W}^{(1)} \in \mathbb{R}^{d \times d_h}, \mathbf{W}^{(2)} \in \mathbb{R}^{d_h \times C}\}$ is the model parameters.

The MLP classification model can be also replaced with more complex and advanced GNN models, including GCN and GAT. The experimental results show that the replacements result in consistent performance drop across different datasets due to GNNs' over-smoothing problem (Cf. Appendix A.5.3 for details).

With this data flow, it can be realized that GRAND actually separates the feature propagation (i.e., $\overline{\mathbf{X}} = \overline{\mathbf{A}}\widetilde{\mathbf{X}}$ in random propagation) and transformation (i.e., $\sigma(\overline{\mathbf{X}}\mathbf{W})$ in prediction) steps, which are coupled with each other in standard GNNs (i.e., $\sigma(\mathbf{AXW})$). This allows us to perform the high-order feature propagation $\overline{\mathbf{A}} = \frac{1}{K+1}\sum_{k=0}^{K}\hat{\mathbf{A}}^k$ without increasing the complexity of neural networks, reducing the risk of overfitting and over-smoothing.

3.2 Consistency Regularized Training

We show that random propagation can be seen as an efficient method for stochastic data augmentation. As such, it is natural to design a consistency regularized training algorithm for GRAND.

Random Propagation as Stochastic Data Augmentation. Random propagation randomly drops some nodes' entire features before propagation. As a result, each node only aggregates information from a random subset of its (multi-hop) neighborhood. In doing so, we are able to stochastically generate different representations

是随机，一个节点邻居节点由mask控制是否存在，
所以对于节点来说其邻居节点组成的子图是随机

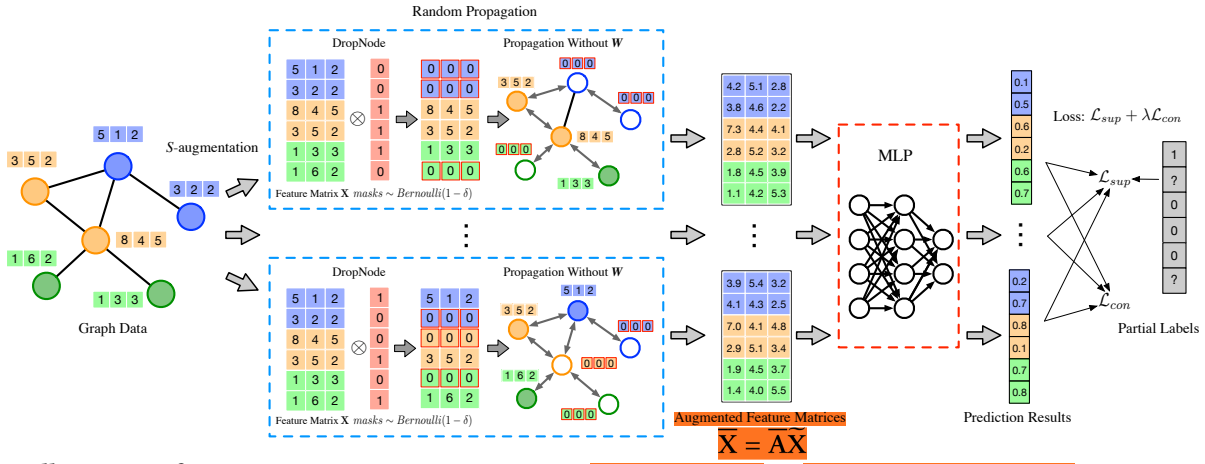


Figure 2: Illustration of GRAND. GRAND consists of two mechanisms—random propagation and consistency regularized training. In random propagation, some nodes' feature vectors are randomly dropped with DropNode. The resultant perturbed feature matrix $\widetilde{\mathbf{X}}$ is then used to perform propagation without parameters to learn. Further, random propagation is used for stochastic graph data augmentation. After that, the augmented feature matrices are fed into a two-layer MLP model for prediction. With applying consistency regularized training, GRAND generates S data augmentations by performing random propagation S times, and leverages both supervised classification loss $\mathcal{L}_{sup}$ and consistency regularization loss $\mathcal{L}_{con}$ in optimization.

因为对称归一化邻接矩阵没有参数 随机图

for each node, which can be considered as a stochastic graph augmentation method. In addition, random propagation can be seen as injecting random noise into the propagation procedure.

To empirically examine this data augmentation idea, we generate a set of augmented node representations $\overline{\mathbf{X}}$ with different drop rates in random propagation and use each $\overline{\mathbf{X}}$ to train a GCN for node classification on commonly used datasets—Cora, Citeseer, and Pubmed. The results show that the decrease in GCN's classification accuracy is less than 3% even when the drop rate is set to 0.5. In other words, with half of rows in the input $\mathbf{X}$ removed (set to $\vec{0}$), random propagation is capable of generating augmented node representations that are sufficient for prediction.

Though one single $\overline{\mathbf{X}}$ is relatively inferior to the original $\mathbf{X}$ in performance, in practice, multiple augmentations—each per epoch—are utilized for training the GRAND model. Similar to bagging [5], GRAND's random data augmentation scheme makes the final prediction model implicitly assemble models on exponentially many augmentations, yielding much better performance than the deterministic propagation used in GCN and GAT.

差的

S–augmentation. Inspired by the above observation, we propose to generate S different data augmentations for the input graph data $\mathbf{X}$. In specific, we perform the random propagation operation for S times to generate S augmented feature matrices $\{\overline{\mathbf{X}}^{(s)} | 1 \le s \le S\}$. Each of these augmented feature matrices is fed into the MLP prediction module to get the corresponding output:

$$\widetilde{\mathbf{Z}}^{(s)} = P(\mathbf{Y}|\overline{\mathbf{X}}^{(s)}; \Theta), \tag{3}$$

where $\widetilde{\mathbf{Z}}^{(s)} \in [0,1]^{n \times C}$ denotes the classification probabilities on the s^{th} augmented data $\overline{\mathbf{X}}^{(s)}$.

Classification Loss. With m labeled nodes among n nodes, the supervised objective of the graph node classification task in each epoch is the average cross-entropy loss over S augmentations:

$$\mathcal{L}_{sup} = -\frac{1}{S}\sum_{s=1}^{S}\sum_{i=0}^{m-1} \mathbf{Y}_i \cdot \log \widetilde{\mathbf{Z}}_i^{(s)}, \tag{4}$$

where $\widetilde{\mathbf{Z}}_i^{(s)}$ is the i^{th} row vector of $\widetilde{\mathbf{Z}}^{(s)}$. Optimizing this loss enforces the model to output the same predictions for (only) labeled nodes on different augmentations. However, labeled data is often very rare in the semi-supervised setting, in which we would like to also make full use of unlabeled data.

Consistency Regularization Loss. In the semi-supervised setting, we propose to optimize the consistency among S augmentations for unlabeled data. Considering a simple case of $S = 2$, we can minimize the distributional distance between the two outputs, i.e., $\min \sum_{i=0}^{n-1} \mathcal{D}(\widetilde{\mathbf{Z}}_i^{(1)}, \widetilde{\mathbf{Z}}_i^{(2)})$, where $\mathcal{D}(\cdot,\cdot)$ is the distance

Z1与Z2分别是增强数据X1与X2对整个图所有节点的分类结果，最小它们之间的距离是有意义的

Algorithm 2 Consistency Regularized Training for GRAND

Input:
Adjacency matrix $\hat{\mathbf{A}}$, feature matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, times of augmentations in each epoch S, DropNode probability δ.
Output:
Prediction $\mathbf{Z}$.

while not convergence **do**
  for $s = 1 : S$ **do**
    Apply DropNode via Algorithm 1: $\widetilde{\mathbf{X}}^{(s)} \sim \text{DropNode}(\mathbf{X}, \delta)$.
    Perform propagation: $\overline{\mathbf{X}}^{(s)} = \frac{1}{K+1}\sum_{k=0}^{K} \hat{\mathbf{A}}^k \widetilde{\mathbf{X}}^{(s)}$.
    Predict class distribution using MLP: $\widetilde{\mathbf{Z}}^{(s)} = P(\mathbf{Y}|\overline{\mathbf{X}}^{(s)}; \Theta)$.
  end for
  Compute supervised classification loss $\mathcal{L}_{sup}$ via Eq. 4 and consistency regularization loss via Eq. 6.
  Update the parameters Θ by gradients descending:

$$\nabla_\Theta \mathcal{L}_{sup} + \lambda \mathcal{L}_{con}$$

end while
Output prediction $\mathbf{Z}$ via Eq. 8.

function. To extend this idea into multiple-augmentation situation, we first calculate the label distribution center by taking the average of all distributions, i.e., $\overline{\mathbf{Z}}_i = \frac{1}{S}\sum_{s=1}^{S}\widetilde{\mathbf{Z}}_i^{(s)}$. Then we minimize the distributional distance between $\widetilde{\mathbf{Z}}_i^{(s)}$ and $\overline{\mathbf{Z}}_i$, i.e., $\min \sum_{s=1}^{S}\sum_{i=0}^{n-1}\mathcal{D}(\overline{\mathbf{Z}}_i, \widetilde{\mathbf{Z}}_i^{(s)})$. However, the distribution center calculated in this way is always inclined to have higher entropy values, indicating greater "uncertainty". Consequently, it will bring extra uncertainty into the model's predictions. To avoid this problem, we utilize the *label sharpening* trick here. Specifically, we apply a sharpening function onto the averaged label distribution to reduce its entropy [3], i.e.,

$$\overline{\mathbf{Z}}'_{ik} = \overline{\mathbf{Z}}_{ik}^{\frac{1}{T}} \Big/ \sum_{j=0}^{C-1}\overline{\mathbf{Z}}_{ij}^{\frac{1}{T}}, \tag{5}$$

where $0 < T \leq 1$ acts as the "temperature" that controls the sharpness of the categorical distribution. As $T \to 0$, the sharpened label distribution will approach a one-hot distribution. To substitute $\overline{\mathbf{Z}}_i$ with $\overline{\mathbf{Z}}'_i$, we minimize the distance between $\widetilde{\mathbf{Z}}_i$ and $\overline{\mathbf{Z}}'_i$ in GRAND:

$$\mathcal{L}_{con} = \frac{1}{S}\sum_{s=1}^{S}\sum_{i=0}^{n-1}\mathcal{D}(\overline{\mathbf{Z}}'_i, \widetilde{\mathbf{Z}}_i^{(s)}). \tag{6}$$

Therefore, by setting T as a small value, we can enforce the model to output low-entropy predictions. This can be viewed as adding an extra entropy minimization regularization into the model, which assumes that the classifier's decision boundary should not pass through high-density regions of the marginal data distribution [21].

As for the distance function $\mathcal{D}(\cdot,\cdot)$, we adopt the squared L_2 loss $\mathcal{D}(\mathbf{a},\mathbf{b}) = \|\mathbf{a}-\mathbf{b}\|^2$ in our model. Recent studies have demonstrated that it is less sensitive to incorrect predictions [3] and thus is more suitable for the semi-supervised setting than cross-entropy.

Semi-supervised Training and Inference. In each epoch, we employ both the supervised classification loss in Eq. 4 and the consistency regularization loss in Eq. 6 on S augmentations. Hence, the final loss of GRAND is:

$$\mathcal{L} = \mathcal{L}_{sup} + \lambda\mathcal{L}_{con}, \tag{7}$$

where λ is a hyper-parameter that controls the balance between the supervised classification and consistency regularization losses.

During inference, as mentioned in Section 3.1, we directly use the original feature $\mathbf{X}$ without DropNode for propagation. This is justified because we scaled the perturbed feature matrix $\widetilde{\mathbf{X}}$ during training to guarantee its expectation to match $\mathbf{X}$. Hence the inference formula is:

$$\mathbf{Z} = P\left(\mathbf{Y}\,\middle|\,\frac{1}{K+1}\sum_{k=0}^{K}\hat{\mathbf{A}}^k\mathbf{X};\hat{\Theta}\right), \tag{8}$$

where $\hat{\Theta}$ denotes the optimized parameters after training. Algorithm 2 outlines GRAND's training process.

3.3 Complexity Analysis

GRAND comprises of random propagation and consistency regularized training. For random propagation, we compute $\overline{\mathbf{X}}$ by iteratively calculating the product of $\hat{\mathbf{A}}^k$ and $\widetilde{\mathbf{X}}$, and its time complexity is $O(Kd(n+|E|))$. The complexity of its prediction module (2-layer MLP) is $O(nd_h(d+C))$, where d_h denotes its hidden size. By applying consistency regularized training, the total computational complexity of GRAND is $O(S(Kd(n+|E|)+nd_h(d+C))$, **which is linear with the sum of node and edge counts.**

4 THEORETICAL ANALYSIS

Dropout在原数据集上创造一个有噪声的环境，让模型能在有噪声环境学习正确识别数据特征的能力，增强了模型的泛化性

We discuss how *random propagation* and *consistency regularization* can help enhance GRAND's generalization. The key idea is to explore the additional effects that DropNode brings on the model optimization. The main theoretical conclusions include:

- The DropNode regularization with consistency regularization (or supervised classification) loss can enforce the consistency of the classification confidence between each node and its all (or labeled) multi-hop neighborhoods.
- Dropout is actually an adaptive L_2 regularization for $\mathbf{W}$ in GNNs, and its regularization term is the upper bound of DropNode's. By minimizing this term, dropout can be regarded as an approximation of DropNode.

4.1 Consistency Regularization Loss

DropNode can be viewed as injecting perturbations to each node by multiplying the scaled Bernoulli random variable with its feature vector, i.e., $\widetilde{\mathbf{X}}_i = \frac{\epsilon_i}{1-\delta}\cdot\mathbf{X}_i$, with ϵ_i drew from $Bernoulli(1-\delta)$. For analytical simplicity, we assume that the MLP used in GRAND has one single output layer, and the task is binary classification.

Thus GRAND's output $\widetilde{\mathbf{Z}} \in \mathbb{R}^n$ in Eq. 2 can be rewritten as: $\widetilde{\mathbf{Z}} = \text{sigmoid}(\overline{\mathbf{A}}\widetilde{\mathbf{X}}\cdot\mathbf{W})$, where $\overline{\mathbf{A}} = \frac{1}{K+1}\sum_{k=0}^{K}\hat{\mathbf{A}}^k$ and $\mathbf{W} \in \mathbb{R}^d$ is the learnable parameter matrix. For the i^{th} node, the corresponding conditional distribution is $P(y_i|\mathbf{A},\mathbf{X},\mathbf{W}) = \tilde{z}_i^{y_i}(1-\tilde{z}_i)^{1-y_i}$, in which $\tilde{z}_i \in \widetilde{\mathbf{Z}} = \text{sigmoid}(\overline{\mathbf{A}}_i\widetilde{\mathbf{X}}\cdot\mathbf{W})$ and $y_i \in \{0,1\}$ denotes the corresponding label.

As for the consistency regularization loss, without loss of generality, we consider the case of performing the random propagation twice, i.e., $S = 2$, and adopt the squared L2 loss as the distance function $\mathcal{D}(\cdot,\cdot)$. Then the consistency regularization loss can be rewritten as:

n代表图中节点总数

$$\mathcal{L}_{con} = \frac{1}{2}\sum_{i=0}^{n-1}\left(\tilde{z}_i^{(1)} - \tilde{z}_i^{(2)}\right)^2, \tag{9}$$

where $\tilde{z}_i^{(1)}$ and $\tilde{z}_i^{(2)}$ represent the model's two outputs on node i corresponding to the two augmentations. With these assumptions, we can prove that:

THEOREM 1. *In expectation, optimizing the unsupervised consistency loss $\mathcal{L}_{con}$ is approximate to optimize a regularization term*: $\mathbb{E}_\epsilon(\mathcal{L}_{con}) \approx \mathcal{R}^c(\mathbf{W}) = \sum_{i=0}^{n-1} z_i^2(1-z_i)^2\text{Var}_\epsilon\left(\overline{\mathbf{A}}_i\widetilde{\mathbf{X}}\cdot\mathbf{W}\right)$.

The proof details can be found in Appendix A.4.1. For DropNode with the drop rate δ, we can easily check that

$$\text{Var}_\epsilon(\overline{\mathbf{A}}_i\widetilde{\mathbf{X}}\cdot\mathbf{W}) = \frac{\delta}{1-\delta}\sum_{j=0}^{n-1}(\mathbf{X}_j\cdot\mathbf{W})^2(\overline{\mathbf{A}}_{ij})^2. \tag{10}$$

Then the regularization term $\mathcal{R}^c$ can be expressed as:

$$\mathcal{R}^c(\mathbf{W}) = \frac{\delta}{1-\delta}\sum_{j=0}^{n-1}\left[(\mathbf{X}_j\cdot\mathbf{W})^2\sum_{i=0}^{n-1}(\overline{\mathbf{A}}_{ij})^2 z_i^2(1-z_i)^2\right]. \tag{11}$$

Note that $z_i(1-z_i)$ (or its square) is an indicator of the classification uncertainty for the i^{th} node, as $z_i(1-z_i)$ (or its square) reaches its maximum at $z_i = 0.5$ and minimum at $z_i = 0$ or 1. Thus

$\sum_{i=0}^{m-1}(\overline{\mathbf{A}}_{ij})^2 z_i^2(1-z_i)^2$ can be viewed as the weighted average classification uncertainty over the j^{th} node's multi-hop neighborhoods with the weights as the square values of $\overline{\mathbf{A}}$'s elements, which is related to graph structure. On the other hand, $(\mathbf{X}_j \cdot \mathbf{W})^2$—as the square of the input of sigmoid—indicates the classification confidence for the j^{th} node. In optimization, in order for a node to earn a higher classification confidence $(\mathbf{X}_j \cdot \mathbf{W})^2$, it is required that the node's neighborhoods have lower classification uncertainty scores.

Hence, *the dropnode regularization with the consistency regularization loss can enforce the consistency of the classification confidence between each node and its all multi-hop neighborhoods.*

4.2 Supervised Classification Loss

We discuss the regularization of dropnode w.r.t the superivised classification loss. We follow the assumption settings expressed in Section 4.1. Then the supervised classification loss is:

$$\mathcal{L}_{sup} = \sum_{i=0}^{m-1} -y_i \log(\tilde{z}_i) - (1-y_i)\log(1-\tilde{z}_i). \tag{12}$$

$\mathcal{L}_{sup}$ refers to the perturbed classification loss with DropNode on the node features. By contrast, the original (non-perturbed) classification loss is defined as

$$\mathcal{L}_{org} = \sum_{i=0}^{m-1} -y_i \log(z_i) - (1-y_i)\log(1-z_i), \tag{13}$$

where $z_i = \text{sigmoid}(\overline{\mathbf{A}}_i\mathbf{X}\cdot\mathbf{W})$ is the output with the original feature matrix $\mathbf{X}$. Then we have the following theorem with proof in Appendix A.4.2.

Theorem 2. *In expectation, optimizing the perturbed classification loss $\mathcal{L}_{sup}$ is equivalent to optimize the original loss $\mathcal{L}_{org}$ with an extra regularization term $\mathcal{R}(\mathbf{W})$, which has a quadratic approximation form $\mathcal{R}(\mathbf{W}) \approx \mathcal{R}^q(\mathbf{W}) = \frac{1}{2}\sum_{i=0}^{m-1} z_i(1-z_i)\text{Var}_\epsilon\left(\overline{\mathbf{A}}_i\widetilde{\mathbf{X}}\cdot\mathbf{W}\right)$.*

This theorem suggests that DropNode brings an extra regularization loss to the optimization objective. Based on Eq. 10, this extra quadratic regularization loss can be expressed as:

$$\mathcal{R}^q(\mathbf{W}) = \frac{1}{2}\frac{\delta}{1-\delta}\sum_{j=0}^{n-1}\left[(\mathbf{X}_j\cdot\mathbf{W})^2\sum_{i=0}^{m-1}(\overline{\mathbf{A}}_{ij})^2 z_i(1-z_i)\right]. \tag{14}$$

Different from $\mathcal{R}^c$ in Eq. 11, the inside summation term in Eq. 14 only incorporates the first m nodes, i.e, the labeled nodes.

Thus, *the dropnode regularization with supervised classification loss can enforce the consistency of the classification confidence between each node and its labeled multi-hop neighborhoods.*

4.3 DropNode vs. Dropout

Dropout [40] is a general regularization method for preventing overfitting in deep learning. It removes each element of $\mathbf{X}$ independently, while DropNode drops a node's entire feature if it is selected. Figure 3 shows their difference.

Formally, dropout perturbs the feature matrix by randomly setting some elements of $\mathbf{X}$ to 0, i.e., $\widetilde{\mathbf{X}}_{ij} = \frac{\tilde{\epsilon}_{ij}}{1-\delta}\mathbf{X}_{ij}$, where $\tilde{\epsilon}_{ij}$ draws from $Bernoulli(1-\delta)$. With this, both Theorems 1 and 2 hold for dropout as well with $\text{Var}_{\tilde{\epsilon}}(\overline{\mathbf{A}}_i\widetilde{\mathbf{X}}\cdot\mathbf{W})$ as:

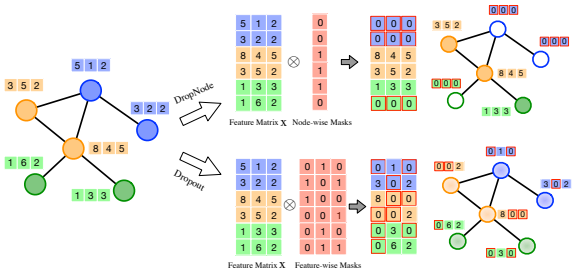


Figure 3: Difference between dropnode and dropout. Dropout drops each element in $\mathbf{X}$ independently, while DropNode drops the entire features of selected nodes, i.e., the row vectors of $\mathbf{X}$, randomly.

Table 1: Benchmark Dataset statistics.

Dataset	Nodes	Edges	Train/Valid/Test Nodes	Classes	Features
Cora	2,708	5,429	140/500/1,000	7	1,433
Citeseer	3,327	4,732	120/500/1,000	6	3,703
Pubmed	19,717	44,338	60/500/1,000	3	500

$$\text{Var}_{\tilde{\epsilon}}(\overline{\mathbf{A}}_i\widetilde{\mathbf{X}}\cdot\mathbf{W}) = \frac{\delta}{1-\delta}\sum_{j=0}^{n-1}\sum_{k=0}^{d-1}\mathbf{X}_{jk}^2\mathbf{W}_k^2(\overline{\mathbf{A}}_{ij})^2. \tag{15}$$

Without loss of generality, we focus on the consistency regularization loss and the corresponding regularization is:

$$\widetilde{\mathcal{R}}^c(\mathbf{W}) = \frac{\delta}{1-\delta}\sum_{h=0}^{d-1}\mathbf{W}_h^2\sum_{j=0}^{n-1}\left[\mathbf{X}_{jh}^2\sum_{i=0}^{n-1}z_i^2(1-z_i)^2(\overline{\mathbf{A}}_{ij})^2\right]. \tag{16}$$

Similar to DropNode, this extra regularization term also includes the classification uncertainty $z_i(1-z_i)$ of neighborhoods. However, *we can observe that different from the DropNode regularization, dropout is actually an adaptive L_2 regularization for $\mathbf{W}$, where the regularization coefficient is associated with unlabeled data, classification uncertainty, and the graph structure.*

By applying the Cauchy-Schwarz Inequality to Eq. 16, we have:

$$\widetilde{\mathcal{R}}^c(\mathbf{W}) \geq \frac{\delta}{1-\delta}\sum_{j=0}^{n-1}\left[(\mathbf{X}_j\cdot\mathbf{W})^2\sum_{i=0}^{n-1}(\overline{\mathbf{A}}_{ij})^2 z_i^2(1-z_i)^2\right] = \mathcal{R}^c(\mathbf{W}) \tag{17}$$

That is to say, dropout's regularization term is the upper bound of DropNode's. By minimizing this term, dropout can be regarded as an approximation of DropNode.

5 EXPERIMENTS

5.1 Experimental Setup

Datasets. We conduct experiments on three benchmark graphs [27, 42, 47]—Cora, Citeseer, and Pubmed—and also report results on six publicly available and relatively large datasets in Appendix A.3. Tables 1 and 5 summarize the dataset statistics, respectively. *We use exactly the same experimental settings—such as features and data splits—on the three benchmark datasets as literature on semi-supervised graph mining [27, 42, 47].*

Baselines. To comprehensively evaluate the Grand model[1], we compare it with 12 state-of-the-art GNNs and 3 Grand variants.

- **GCN** [27] uses the propagation rule described in Eq. 1.

[1]The code has been published in https://github.com/Grand20/grand.

- **GAT** [42] propagates information based on self-attention.
- **Graph U-Net** [15] proposes the graph pooling operations.
- **MixHop** [2] employs the mixed-order propagation.
- **GMNN** [36] combines GNNs with probabilistic graphical models.
- **GraphNAS** [16] automatically generates GNN architectures using reinforcement learning.
- **VBAT** [13] applies virtual adversarial training [33] into GCNs.
- **G^{3}NN** [31] regularizes GNNs with an extra link prediction task.
- **GraphMix** [43] adopts MixUp [49] for regularizing GNNs.
- **DropEdge** [37] randomly drops some edges in GNNs training.
- **GraphSAGE** [22] proposes node-wise neighborhoods sampling.
- **FastGCN** [9] using importance sampling for fast GCNs training.
- **Grand_GCN**. Note that the prediction module in Grand is the simple MLP model. In Grand_GCN, we replace MLP with GCN.
- **Grand_GAT** replaces the MLP component in Grand with GAT.
- **Grand_dropout** substitutes our DropNode technique with the dropout operation in Grand's random propagation.

5.2 Overall Results

Table 2 summarizes the prediction accuracies of node classification. Following the community convention [27, 36, 42], the results of baseline models are taken from the corresponding works [2, 13, 15, 16, 27, 31, 37, 42, 43]. The accuracy results of our methods in Table 2 are averaged over **100 runs** with random weight initializations.

From Table 2, we can observe that Grand consistently achieves large-margin outperformance over all baselines across all datasets. Specifically, Grand improves upon GCN by a margin of 3.9%, 5.1%, and 3.7% (absolute differences) on Cora, Citeseer, and Pubmed, while the margins improved by GAT upon GCN were 1.5%, 2.2%, and 0%, respectively. When compared to the very recent regularization based model—DropEdge, the proposed model achieves 2.6%, 3.1%, and 3.1% improvements, while DropEdge's improvements over GCN were only 1.3%, 2.0%, and 0.6%, respectively.

We interpret the performance of Grand_GAT, Grand_GCN, and Grand_dropout from three perspectives. First, both Grand_GAT and Grand_GCN outperform the original GCN and GAT models, demonstrating the positive effects of the proposed random propagation and consistency regularized training methods. Second, both of them are inferior to Grand with the simple MLP model, suggesting GCN and GAT are relatively easier to over-smooth than MLP. Detailed analyses on this can be found in Appendix A.5.3. Finally, we observe a clear performance drop when replacing DropNode with dropout (Grand_dropout vs. Grand), demonstrating that the DropNode technique is more suitable than dropout for graph data.

Detailed experiments to compare DropNode and dropout under different propagation steps K are shown in Appendix A.5.1.

5.3 Ablation Study

In addition to analyze Grand_GAT, Grand_GCN, and Grand_dropout, we conduct an ablation study to examine the contribution of different components by removing them:

- **Without CR:** Remove consistency regularization (CR), i.e., $\lambda = 0$, meaning that we only use supervised classification loss.

[2]We report the results of these methods with GCN as the backbone model.
[3]The experiments of FastGCN and GraphSAGE are conducted by ourselves under transductive semi-supervised setting with 100 trials.

Table 2: Summary of classification accuracy (%).

Category	Method	Cora	Citeseer	Pubmed
Graph Convolution	GCN [27]	81.5	70.3	79.0
	GAT [42]	83.0±0.7	72.5±0.7	79.0 ±0.3
	Graph U-Net [15]	84.4±0.6	73.2±0.5	79.6±0.2
	MixHop [2]	81.9 ± 0.4	71.4±0.8	80.8±0.6
	GMNN [36]	83.7	72.9	81.8
	GraphNAS [16]	84.2±1.0	73.1±0.9	79.6±0.4
Regularization based GCNs[2]	VBAT [13]	83.6 ± 0.5	74.0 ± 0.6	79.9 ± 0.4
	G^{3}NN [31]	82.5 ±0.2	74.4±0.3	77.9 ± 0.4
	GraphMix [43]	83.9 ± 0.6	74.5 ±0.6	81.0 ± 0.6
	DropEdge [37]	82.8	72.3	79.6
Sampling based GCNs[3]	GraphSAGE [22]	78.9±0.8	67.4±0.7	77.8±0.6
	FastGCN [9]	81.4±0.5	68.8±0.9	77.6±0.5
Our methods	Grand	**85.4±0.4**	**75.4±0.4**	**82.7±0.6**
	Grand_GCN	84.5±0.3	74.2±0.3	80.0±0.3
	Grand_GAT	84.3±0.4	73.2± 0.4	79.2±0.6
	Grand_dropout	84.9±0.4	75.0±0.3	81.7±1.0

Table 3: Ablation study results (%).

Model	Cora	Citeseer	Pubmed
Grand	**85.4±0.4**	**75.4±0.4**	**82.7±0.6**
without CR	84.4 ±0.5	73.1 ±0.6	80.9 ±0.8
without multiple DropNode	84.7 ±0.4	74.8±0.4	81.0±1.1
without sharpening	84.6 ±0.4	72.2±0.6	81.6 ± 0.8
without CR and DropNode	83.2 ± 0.5	70.3 ± 0.6	78.5± 1.4

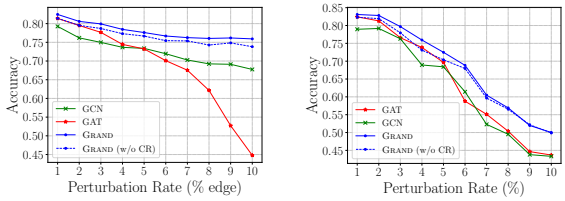
- **Without multiple DropNode:** Do DropNode once at each epoch, i.e., $S = 1$, meaning that CR only enforces the model to give low-entropy predictions for unlabeled nodes.
- **Without sharpening:** Do not use the label sharpening trick (Cf. Eq. 5) in calculating the distribution center for CR, i.e., $T = 1$.
- **Without DropNode (and CR):** Remove DropNode (as a result, the CR loss is also removed), i.e., $\delta = 0, \lambda = 0$. In this way, Grand becomes the combination of deterministic propagation and MLP.

Table 3 summarizes the results of the ablation study, from which we have the following two observations. First, all Grand variants with some components removed witness clear performance drops when comparing to the full model, suggesting that each of the designed components contributes to the success of Grand. Second, Grand without consistency regularization outperforms almost all six non-regularization based GCNs in all three datasets, demonstrating the significance of the proposed random propagation technique for semi-supervised graph learning.

5.4 Addressing Non-robustness, Over-smoothing and Overfitting

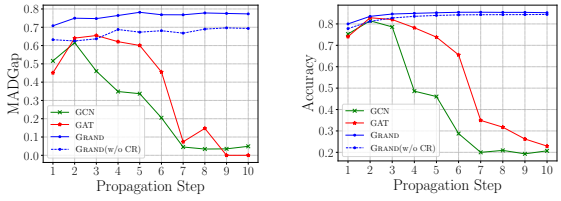
Robustness Analysis. We study the robustness of the proposed Grand model. Specifically, we utilize the following adversarial attack methods to generate perturbed graphs, and then examine the model's classification accuracies on them.

- **Random.** Perturbing the structure by randomly adding fake edges.
- **Metattack [54].** Attacking the graph structure by removing or adding edges based on meta learning.



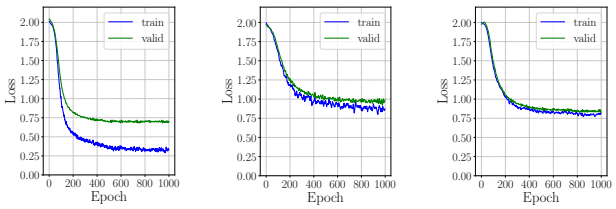
(a) Random Attack (b) Metattack

Figure 4: Robustness: results under attacks on Cora.



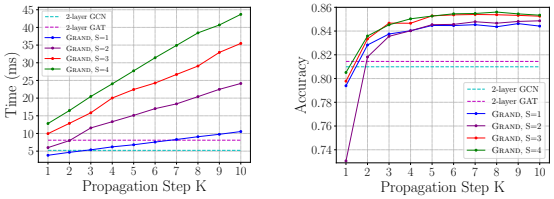
(a) MADGap (b) Classification Results

Figure 5: Over-smoothing: GRAND vs. GCN & GAT on Cora.



(a) Without CR and RP (b) Without CR (c) GRAND

Figure 6: Generalization: Training/validation losses on Cora.



(a) Per-epoch Training Time (b) Classification Accuracy

Figure 7: Efficiency Analysis for GRAND.

Figure 4 presents the classification accuracies of different methods with respect to different perturbation rates on the Cora dataset. We observe that GRAND consistently outperforms GCN and GAT across all perturbation rates on both attacks. When adding 10% new random edges into Cora, we observe only a 7% drop in classification accuracy for GRAND, while 12% for GCN and 37% for GAT. Under Metattack, the gap between GRAND and GCN/GAT also enlarges with the increase of the perturbation rate. This study suggests the robustness advantage of the GRAND model (with or without) consistency regularization over GCN and GAT.

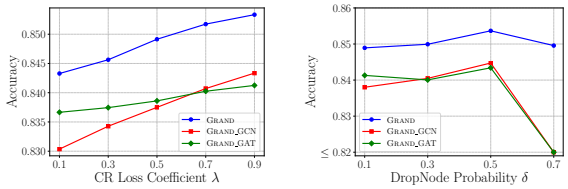
Relieving Over-smoothing. Many GNNs face the over-smoothing issue—*nodes with different labels become indistinguishable*—when enlarging the feature propagation step [8, 29]. We quantitatively study how GRAND is vulnerable to this issue by using MADGap [8]. MADGap measures the over-smoothness of node representations—the cosine distance between the remote and neighboring nodes, wherein the remote nodes are defined as those with different labels, and the neighboring nodes are those with the same labels. A smaller MADGap value indicates the more indistinguishable node representations and thus a more severe over-smoothing issue. Figure 5 shows both the MADGap values of the last layer's representations and classification results of each GNN model with respect to different propagation steps. In GRAND, the propagation step is controlled by the hyperparameter K, while for GCN and GAT, we adjust the propagation step by stacking different hidden layers. The plots suggest that as the propagation step increases, both metrics of GCN and GAT decrease dramatically—MADGap drops from ~0.5 to 0 and accuracy drops from 0.75 to 0.2—due to the over-smoothing issue. However, GRAND behaves completely different: both the performance and MADGap benefit from more propagation steps. This indicates that GRAND is much more powerful to relieve over-smoothing, when existing representative GNNs are very vulnerable to it.

Generalization Improvement. We examine how the proposed techniques—random propagation and consistency regularization—improve the model's generalization capacity. To achieve this, we analyze the model's cross-entropy losses on both training and validation sets on Cora. A small gap between the two losses indicates a model with good generalization. Figure 6 reports the results for GRAND and its two variants. We can observe the significant gap between the validation and training losses when without both consistency regularization (CR) and random propagation (RP), indicating an obvious overfitting issue. When applying only the random propagation (without CR), the gap becomes much smaller. Finally, when further adding the CR loss to make it the full GRAND model, the validation loss becomes much closer to the training loss and both of them are also more stable. This observation demonstrates both the random propagation and consistency regularization can significantly improve GRAND's generalization capability.

5.5 Efficiency and Parameter Analysis

Efficiency Analysis. The efficiency of GRAND is mainly influenced by two hyperparameters: the propagation step K and augmentation times S. Figure 7 reports the average per-epoch training time and classification accuracy of GRAND on Cora under different values of K and S with #training epochs fixed to 1000. It also includes the results of the two-layer GCN and two-layer GAT with the same learning rate, #training epochs and hidden layer size as GRAND.

From Figure 7, we can see that when $K = 2, S = 1$, GRAND outperforms GCN and GAT in terms of both efficiency and effectiveness. In addition, we observe that increasing K or S can significantly improve the model's classification accuracy at the cost of its training efficiency. In practice, we can adjust the values of K and S to balance the trade-off between performance and efficiency.



(a) CR loss coefficient λ (b) DropNode probability δ

Figure 8: Parameter sensitivity of λ and δ on Cora.

Parameter Sensitivity. We investigate the sensitivity of consistency regularization (CR) loss coefficient λ and DropNode probability δ in GRAND and its variants on Cora. The results are shown in Figure 8. We observe that their performance increase when enlarging the value of λ. As for DropNode probability, GRAND, GRAND_GCN and GRAND_GAT reach their peak performance at $\delta = 0.5$. This is because the augmentations produced by random propagation in that case are more stochastic and thus make GRANDgeneralize better with the help of consistency regularization.

6 CONCLUSIONS

In this work, we propose the GRAPH RANDOM NEURAL NETWORKS (GRAND) for semi-supervised learning on graphs. Unlike traditional GNNs that follow a recursive deterministic neighborhood aggregation process, we propose the DropNode technique for random propagation. This operation enables GRAND to generate data augmentations efficiently, which further facilitates the consistency regularization training for semi-supervised graph learning. We demonstrate its significant outperformance over twelve state-of-the-art GNN baselines on benchmark datasets. More importantly, we theoretically illustrate its properties and empirically demonstrate its advantages over conventional GNNs in terms of robustness, resistance to over-smoothing and overfitting, and generalization.

A APPENDIX

A.1 Implementation Details

We make use of PyTorch to implement GRAND and its variants. The random propagation procedure is efficiently implemented with sparse-dense matrix multiplication. The codes of GCN and GRAND_GCN are implemented referring to the PyTorch version of GCN [4]. As for GRAND_GAT and GAT, we adopt the implementation of GAT layer from the PyTorch-Geometric library [5] in our experiments. The weight matrices of classifier are initialized with Glorot normal initializer [18]. We employ Adam [26] to optimize parameters of the proposed methods and adopt early stopping to control the training epochs based on validation loss. Apart from DropNode used in random propagation, we also apply dropout [40] on the input layer and hidden layer of the prediction module used in GRAND and its variants as a common practice of preventing overfitting in optimizing neural network. For the experiments on Pubmed, we also use batch normalization [25] to stabilize the training procedure. All the experiments in this paper are conducted on a single NVIDIA GeForce RTX 2080 Ti with 11 GB memory size. Server operating system is Unbuntu 18.04. As for software versions, we use Python 3.7.3, PyTorch 1.2.0, NumPy 1.16.4, SciPy 1.3.0, CUDA 10.0.

A.2 Hyperparameter Details

Overall Results in Section 5.2. GRAND introduces five additional hyperparameters, that is the DropNode probability δ, propagation step K in random propagation, data augmentation times S at each training epoch, sharpening temperature T when calculating consistency regularization loss and the coefficient of consistency regularization loss λ trading-off the balance between $\mathcal{L}_{sup}$ and $\mathcal{L}_{con}$. In practice, δ is always set to 0.5 across all experiments. As for other hyperparameters, we perform hyperparameter search for each dataset. Specifically, we first search K from { 2,4,5,6,8}. With the best selection of K, we then search S from {2,3,4}. Finally, we fix K and S to the best values and take a grid search for T and λ from {0.1, 0.2, 0.3,0.5} and {0.5, 0.7, 1.0} respectively. For each search of hyperparameter configuration, we run the experiments with 20 random seeds and select the best configuration of hyperparameters based on average accuracy on validation set. Other hyperparameters used in our experiments includes learning rate of Adam, early stopping patience, L2 weight decay rate, hidden layer size, dropout rates of input layer and hidden layer. We didn't spend much effort to tune these hyperparameters in practice, as we observe that GRAND is not very sensitive with those. Table 4 reports the best hyperparameters of GRAND we used for the results reported in Table 2.

Table 4: Hyperparameters of GRAND for results in Table 2.

Hyperparameter	Cora	Citeseer	Pubmed
DropNode probability δ	0.5	0.5	0.5
Propagation step K	8	2	5
Data augmentation times S	4	2	4
CR loss coefficient λ	1.0	0.7	1.0
Sharpening temperature T	0.5	0.3	0.2
Learning rate	0.01	0.01	0.2
Early stopping patience	200	200	100
Hidden layer size	32	32	32
L2 weight decay rate	5e-4	5e-4	5e-4
Dropout rate in input layer	0.5	0.0	0.6
Dropout rate in hidden layer	0.5	0.2	0.8

Robustness Analysis in Section 5.4. For random attack, we implement the attack method with Python and NumPy library. The propagation step K of GRAND (with or without CR) is set to 5. And the other hyperparameters are set to the values in Table 4. As for Metattack [54], we use the publicly available implementation[6] published by the authors with the same hyperparameters used in the original paper. We observe GRAND (with or without CR) is sensitive to the propagation step K under different perturbation rates. Thus we search K from {5,6,7,8} for each perturbation rate. The other hyperparameters are fixed to the values reported in Table 4.

Other Experiments. For the other results reported in Section 5.3 -5.5, the hyperparameters used in GRAND are set to the values reported in Table 4 with one or two changed for the corresponding analysis.

[4] https://github.com/tkipf/pygcn
[5] https://pytorch-geometric.readthedocs.io
[6] https://github.com/danielzuegner/gnn-meta-attack

Baseline Methods. For the results of GCN or GAT reported in Section 5.4-5.5, the learning rate is set to 0.01, early stopping patience is 100, L2 weight decay rate is 5e-4, dropout rate is 0.5. The hidden layer size of GCN is 32. For GAT, the hidden layer consists 8 attention heads and each head consists 8 hidden units.

A.3 Datasets Details

There are totally *nine* datasets used in this paper, that is, Cora, Citeseer, Pubmed, Cora-Full, Coauthor CS, Coauthor Physics, Amazon Computers, Amazon Photo and Aminer CS. The statistics of Cora, Citeseer and Pubmed have been presented in Table 1. Our preprocessing scripts for Cora, Citeseer and Pubmed is implemented with reference to the codes of Planetoid[7]. Following the experimental setup used in [27, 42, 47], we run 100 trials with 100 random seeds for all results on Cora, Citeseer and Pubmed reported in Section 5.

Table 5: Statistics of Large Datasets.

	Classes	Features	Nodes	Edges
Cora-Full	67	8,710	18,703	62,421
Coauthor CS	15	6,805	18,333	81,894
Coauthor Physics	5	8,415	34,493	247,962
Amazon Computers	10	767	13,381	245,778
Amazon Photo	8	745	7,487	119,043
AMiner CS	18	100	593,486	6,217,004

We also evaluate our methods on six relatively large datasets, i.e., Cora-Full, Coauthor CS, Coauthor Physics, Amazon Computers, Amazon Photo and Aminer CS. The statistics of these datasets are given in Table 5. Cora-Full is proposed in [4]. Coauthor CS, Coauthor Physics, Amazon Computers and Amazon Photo are proposed in [39]. We download the processed versions of the five datasets here[8]. AMiner CS is conducted by ourselves based on AMiner citation network[41]. In AMiner CS, each node corresponds to a paper in computer science, and edges represent citation relations between papers. These papers are manually categorized into 18 topics based on their publication venues. We use averaged GLOVE-100 [35] word vector of paper abstract as the node feature vector. Our goal is to predict the corresponding topic of each paper based on feature matrix and citation graph structure. The corresponding results on the six datasets are introduced in Appendix A.3.

A.4 Theorem Proofs

A.4.1 *Proof for Theorem 1 .*

Proof. The expectation of $\mathcal{L}_{con}$ is:

$$\frac{1}{2}\sum_{i=0}^{n-1}\mathbb{E}\left[(\tilde{z}_i^{(1)}-\tilde{z}_i^{(2)})^2\right]=\frac{1}{2}\sum_{i=0}^{n-1}\mathbb{E}\left[\left((\tilde{z}_i^{(1)}-z_i)-(\tilde{z}_i^{(2)}-z_i)\right)^2\right]. \quad (18)$$

Here $z_i=\text{sigmoid}(\overline{\mathbf{A}}_i\mathbf{X}\cdot\mathbf{W})$, $\tilde{z}_i=\text{sigmoid}(\overline{\mathbf{A}}_i\widetilde{\mathbf{X}}\cdot\mathbf{W})$. For the term of $\tilde{z}_i-z_i$, we can approximate it with its first-order Taylor expansion around $\overline{\mathbf{A}}_i\mathbf{X}\cdot\mathbf{W}$, i.e., $\tilde{z}_i-z_i\approx z_i(1-z_i)(\overline{\mathbf{A}}_i(\widetilde{\mathbf{X}}-\mathbf{X})\cdot\mathbf{W})$. Applying this rule to the above equation, we have:

$$\begin{aligned}\frac{1}{2}\sum_{i=0}^{n-1}\mathbb{E}\left[(\tilde{z}_i^{(1)}-\tilde{z}_i^{(2)})^2\right]&\approx\frac{1}{2}\sum_{i=0}^{n-1}z_i^2(1-z_i)^2\mathbb{E}\left[(\overline{\mathbf{A}}_i(\widetilde{\mathbf{X}}^{(1)}-\widetilde{\mathbf{X}}^{(2)})\cdot\mathbf{W})^2\right]\\&=\sum_{i=0}^{n-1}z_i^2(1-z_i)^2\text{Var}_\epsilon\left(\overline{\mathbf{A}}_i\widetilde{\mathbf{X}}\cdot\mathbf{W}\right).\end{aligned} \quad (19)$$

□

A.4.2 *Proof for Theorem 2.*

Proof. Expanding the logistic function, $\mathcal{L}_{org}$ is rewritten as:

$$\mathcal{L}_{org}=\sum_{i=0}^{m-1}\left[-y_i\overline{\mathbf{A}}_i\mathbf{X}\cdot\mathbf{W}+\mathcal{A}(\overline{\mathbf{A}}_i,\mathbf{X})\right], \quad (20)$$

where $\mathcal{A}(\overline{\mathbf{A}}_i,\mathbf{X})=-\log\left(\frac{\exp(-\overline{\mathbf{A}}_i\mathbf{X}\cdot\mathbf{W})}{1+\exp(-\overline{\mathbf{A}}_i\mathbf{X}\cdot\mathbf{W})}\right)$. Then the expectation of perturbed classification loss can be rewritten as:

$$\mathbb{E}_\epsilon(\mathcal{L}_{sup})=\mathcal{L}_{org}+\mathcal{R}(\mathbf{W}), \quad (21)$$

where $\mathcal{R}(\mathbf{W})=\sum_{i=0}^{m-1}\mathbb{E}_\epsilon\left[\mathcal{A}(\overline{\mathbf{A}}_i,\widetilde{\mathbf{X}})-\mathcal{A}(\overline{\mathbf{A}}_i,\mathbf{X})\right]$. Here $\mathcal{R}(\mathbf{W})$ acts as a regularization term for $\mathbf{W}$. To demonstrate that, we can take a second-order Taylor expansion of $\mathcal{A}(\overline{\mathbf{A}}_i,\widetilde{\mathbf{X}})$ around $\overline{\mathbf{A}}_i\mathbf{X}\cdot\mathbf{W}$:

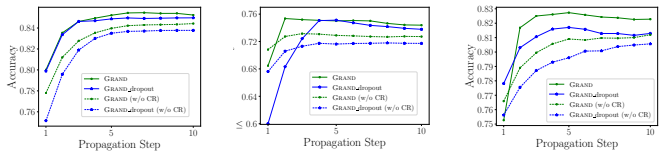
$$\mathbb{E}_\epsilon\left[\mathcal{A}(\overline{\mathbf{A}}_i,\widetilde{\mathbf{X}})-\mathcal{A}(\overline{\mathbf{A}}_i,\mathbf{X})\right]\approx\frac{1}{2}\mathcal{A}''(\overline{\mathbf{A}}_i,\mathbf{X})\text{Var}_\epsilon\left(\overline{\mathbf{A}}_i\mathbf{X}\cdot\mathbf{W}\right). \quad (22)$$

Note that the first-order term $\mathbb{E}_\epsilon\left[\mathcal{A}'(\overline{\mathbf{A}}_i,\mathbf{X})(\widetilde{\mathbf{X}}-\mathbf{X})\right]$ vanishes since $\mathbb{E}_\epsilon(\widetilde{\mathbf{X}})=\mathbf{X}$. We can easily check that $\mathcal{A}''(\overline{\mathbf{A}}_i,\mathbf{X})=z_i(1-z_i)$. Applying this quadratic approximation to $\mathcal{R}(\mathbf{W})$, we get the quadratic approximation form of $\mathcal{R}(\mathbf{W})$:

$$\mathcal{R}(\mathbf{W})\approx\mathcal{R}^q(\mathbf{W})=\frac{1}{2}\sum_{i=0}^{m-1}z_i(1-z_i)\text{Var}_\epsilon(\overline{\mathbf{A}}_i\mathbf{X}\cdot\mathbf{W}). \quad (23)$$

□

A.5 Additional Experiments

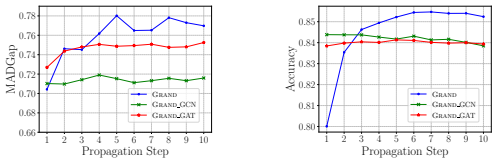


(a) Cora (b) Citeseer (c) Pubmed

Figure 9: GRAND vs. GRAND_dropout.

A.5.1 DropNode vs Dropout. We compare GRAND and GRAND_dropout under different values of propagation step K. The results on Cora, Citeseer and Pubmed are illustrated in Figure 9. We observe GRAND always achieve better performance than GRAND_dropout, suggesting *DropNode is much more suitable for graph data.*

[7] https://github.com/kimiyoung/planetoid
[8] https://github.com/shchur/gnn-benchmark



(a) MADGap (b) Classification Results

Figure 10: Over-smoothing: GRAND vs. its variants on Cora.

A.5.2 Results on Large Datasets. To test GRAND's scalability, we evaluate GRAND on Cora-Full, Coauthor CS, Coauthor Physics, Amazon Computer, Amazon Photo and Aminer CS. Following the evaluation protocol used in [39], we run each model on 100 random train/validation/test splits and 20 random initializations for each split (with **2000** runs on each dataset in total). For each trial, we choose 20 samples for training, 30 samples for validation and the remaining samples for test. We ignore 3 classes with less than 50 nodes in Cora-Full dataset as done in [39]. The results are presented in Table 6. The results of GCN and GAT on the first five datasets are taken from [39]. We can observe that *GRAND significantly outperforms GCN and GAT on all these datasets.*

Table 6: Results on large datasets.

Method	Cora Full	Coauthor CS	Coauthor Physics	Amazon Computer	Amazon Photo	AMiner CS
GCN	62.2 ± 0.6	91.1 ± 0.5	92.8 ± 1.0	82.6 ± 2.4	91.2 ± 1.2	49.9 ± 2.0
GAT	51.9 ± 1.5	90.5 ± 0.6	92.5 ± 0.9	78.0 ± 19.0	85.7 ± 20.3	49.6 ± 1.7
GRAND	**63.5 ±0.6**	**92.9 ± 0.5**	**94.6 ± 0.5**	**85.7 ± 1.8**	**92.5 ± 1.7**	**52.8 ± 1.2**

A.5.3 GRAND vs. GRAND_GCN and GRAND_GAT. As shown in Table 2, GRAND_GCN and GRAND_GAT get worse performances than GRAND, indicating GCN and GAT perform worse than MLP under the framework of GRAND. Here we conduct a series of experiments to analyze the underlying reasons. Specifically, we compare the MADGap values and accuracies GRAND, GRAND_GCN and GRAND_GAT under different values of propagation step K with other parameters fixed. The results are shown in Figure 10. We find that the MADGap and classification accuracy of GRAND increase significantly when enlarging the value of K. However, both the metrics of GRAND_GCN and GRAND_GAT have little improvements or even decrease. This indicates that *GCN and GAT have higher over-smoothing risk than MLP.*

REFERENCES

[1] Sami Abu-El-Haija, Amol Kapoor, Bryan Perozzi, and Joonseok Lee. 2018. N-GCN: Multi-scale graph convolution for semi-supervised node classification. *arXiv preprint arXiv:1802.08888* (2018).

[2] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Hrayr Harutyunyan, Nazanin Alipourfard, Kristina Lerman, Greg Ver Steeg, and Aram Galstyan. 2019. Mixhop: Higher-order graph convolution architectures via sparsified neighborhood mixing. *ICML'19* (2019).

[3] David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin Raffel. 2019. Mixmatch: A holistic approach to semi-supervised learning. *NeurIPS'19* (2019).

[4] Aleksandar Bojchevski and Stephan Günnemann. 2017. Deep Gaussian Embedding of Graphs: Unsupervised Inductive Learning via Ranking. In *ICLR*.

[5] Leo Breiman. 1996. Bagging predictors. *Machine learning* (1996).

[6] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2013. Spectral networks and locally connected networks on graphs. *arXiv:1312.6203* (2013).

[7] Olivier Chapelle, Bernhard Scholkopf, and Alexander Zien. 2009. Semi-supervised learning. *IEEE Transactions on Neural Networks* (2009).

[8] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. 2020. Measuring and Relieving the Over-smoothing Problem for Graph Neural Networks from the Topological View. In *AAAI'20*.

[9] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: fast learning with graph convolutional networks via importance sampling. *arXiv:1801.10247* (2018).

[10] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks. In *KDD'19*.

[11] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. 2018. Adversarial attack on graph structured data. *ICML'18* (2018).

[12] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NeurIPS'16*.

[13] Zhijie Deng, Yinpeng Dong, and Jun Zhu. 2019. Batch Virtual Adversarial Training for Graph Convolutional Networks. *arXiv preprint arXiv:1902.09192* (2019).

[14] Terrance DeVries and Graham W Taylor. 2017. Dataset augmentation in feature space. *ICLR'17* (2017).

[15] Hongyang Gao and Shuiwang Ji. 2019. Graph U-Nets. *ICML'19* (2019).

[16] Yang Gao, Hong Yang, Peng Zhang, Chuan Zhou, and Yue Hu. 2019. GraphNAS: Graph Neural Architecture Search with Reinforcement Learning. *arXiv preprint arXiv:1904.09981* (2019).

[17] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. *arXiv:1704.01212* (2017).

[18] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS'10*.

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press.

[20] Marco Gori, Gabriele Monfardini, and Franco Scarselli. 2005. A new model for learning in graph domains. In *IJCNN'05*.

[21] Yves Grandvalet and Yoshua Bengio. 2005. Semi-supervised learning by entropy minimization. In *NeurIPS'05*.

[22] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS'17*. 1025–1035.

[23] Mikael Henaff, Joan Bruna, and Yann LeCun. 2015. Deep convolutional networks on graph-structured data. *arXiv:1506.05163* (2015).

[24] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. 2018. Adaptive Sampling Towards Fast Graph Representation Learning. In *NeurIPS'18*.

[25] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ICML'15* (2015).

[26] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *ICLR'14* (2014).

[27] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv:1609.02907* (2016).

[28] Samuli Laine and Timo Aila. 2017. Temporal ensembling for semi-supervised learning. *ICLR'17* (2017).

[29] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI'18*.

[30] Leman Akoglu Lingxiao Zhao. 2020. PairNorm: Tackling Oversmoothing in GNNs. *ICLR'20* (2020).

[31] Jiaqi Ma, Weijing Tang, Ji Zhu, and Qiaozhu Mei. 2019. A Flexible Generative Framework for Graph-based Semi-supervised Learning. *NeurIPS'19* (2019).

[32] Yao Ma, Suhang Wang, Charu C. Aggarwal, and Jiliang Tang. 2019. Graph Convolutional Networks with EigenPooling. In *KDD'19*.

[33] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, Ken Nakae, and Shin Ishii. 2016. Distributional smoothing with virtual adversarial training. *ICLR'16* (2016).

[34] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. 2017. Geometric deep learning on graphs and manifolds using mixture model CNNs. In *CVPR'17*.

[35] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global Vectors for Word Representation. In *EMNLP'14*.

[36] Meng Qu, Yoshua Bengio, and Jian Tang. 2019. GMNN: Graph Markov Neural Networks. *ICML'19* (2019).

[37] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. 2020. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. *ICLR'20* (2020).

[38] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2009. The graph neural network model. *IEEE Transactions on Neural Networks* 20, 1 (2009), 61–80.

[39] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868* (2018).

[40] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *JMLR* (2014).

[41] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. Arnetminer: extraction and mining of academic social networks. In *KDD'08*.

[42] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. *ICLR'18* (2018).

[43] Vikas Verma, Meng Qu, Alex Lamb, Yoshua Bengio, Juho Kannala, and Jian Tang. 2019. GraphMix: Regularized Training of Graph Neural Networks for Semi-Supervised Learning. arXiv:1909.11715

[44] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. 2008. Extracting and composing robust features with denoising autoencoders. In *ICML'08*.

[45] Hongwei Wang, Fuzheng Zhang, Mengdi Zhang, Jure Leskovec, Miao Zhao, Wenjie Li, and Zhongyuan Wang. 2019. Knowledge-Aware Graph Neural Networks with Label Smoothness Regularization for Recommender Systems. In *KDD'19*.

[46] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How Powerful are Graph Neural Networks? *arXiv:1810.00826* (2018).

[47] Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. 2016. Revisiting semi-supervised learning with graph embeddings. *ICML'16* (2016).

[48] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *KDD'18*.

[49] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. 2018. mixup: Beyond empirical risk minimization. *ICLR'18* (2018).

[50] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. In *NeurIPS'18*.

[51] Dingyuan Zhu, Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2019. Robust Graph Convolutional Networks Against Adversarial Attacks. In *KDD'19*.

[52] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. 2019. Layer-Dependent Importance Sampling for Training Deep and Large Graph Convolutional Networks. In *NeurIPS'19*.

[53] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. 2018. Adversarial attacks on neural networks for graph data. In *KDD'18*.

[54] Daniel Zügner and Stephan Günnemann. 2019. Adversarial attacks on graph neural networks via meta learning. *ICLR'19* (2019).