Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Understanding and Visualizing Graph Neural Networks

Semester Project

Yifan Lu

yiflu@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Lukas Faber
Prof. Dr. Roger Wattenhofer

September 21, 2020

Acknowledgements

I would like to thank my supervisor Lukas Faber for his dedicated support and guidance. Lukas continuously provided encouragement and was always willing and enthusiastic to assist in any way he could despite his busy schedules. I would also like to thank Prof. Dr. Roger Wattenhofer who gave me this opportunity to undertake this project on the topic of GNNs, so that I could be exposed to this rapidly developing research area. Finally, many thanks to all labmates that have provided advice for this project.

Abstract

Graph neural networks (GNNs) have achieved unparalleled results on tasks related to graph-structured data. Despite their massive practical success, understanding the mechanism and limitation of GNNs is still a crucial task. In this paper[1], we attempt to gain a better understanding of GNNs from three different angles. On account of the over-smoothing issue GNNs face, we propose to train GNNs with an additional loss function to find the fixed points of node embeddings and thus learn more robust representations. Our proposed method achieves competitive results as other baseline methods. Besides, preserving feature identity is arguably a desirable property for many neural network architectures. To investigate whether nodes retain their identity across GNN layers, we borrow the concept "identifiability" from [1] and adapt it to GNNs. Our experiment result reveals that certain nodes retain their identity across the model, however, identifiability does not impact the performance of GNNs. Finally, we take a step back and measure the demands of using GNNs by introducing a new concept *Graph Neural Network Necessity* and then assess several most frequently used node classification datasets according to it.

[1]The code is available at `https://github.com/wwwfan628/gnn_visualization`

Contents

Chapter 1

Introduction

Graphs are rich, flexible and universal structures existing in many high-impact real-world applications such as social networks [2, 3, 4, 5], biological networks [6, 7, 8, 9] and chemical molecules [10, 11, 12]. Graph-type data used to be tackled predominantly with graph kernels methods. Despite massive success, graph kernels methods suffer significant limitations, like high time complexity, disjoint feature and rule learning [5]. The recent advances of optimization techniques and great progresses in parallel computation have made *Graph Neural Network* a compelling alternative.

Various promising GNNs methods were proposed in the last decade [13, 3, 4, 14] and many of them boosted performance of graph related tasks, such as node classification [15, 2, 3, 4], graph classification [12, 16, 13, 17] and link prediction [18, 19, 20, 21]. GNNs generalize deep learning architectures on grid-structured data to graph-type data. Typically, they consist of *feature aggregation* and *graph-level readout*. During the feature aggregation, node representations in a low-dimensional feature space are learned by aggregating and transforming feature vectors from a node's local neighborhood [3, 22, 4]. Representation of entire graph are then obtained using graph-level pooling schemes [18, 13] that compress node representations into a global feature vector.

Compared with their practical popularity, theoretical research and interpretation of GNNs have not been explored extensively. The current rapid growth of GNNs calls for a deeper insight into the internal behavior of these complex models and a better understanding on how decisions are made by them. In this paper, we focus on understanding GNNs from the following three aspects:

- We attempt to train GNNs to find fixed points of node embeddings with a joint loss function consisting of the normal classification loss and an additional loss function that seeks to minimize the difference between input and output of last hidden layer. These fixed points are a step forward towards more stable representations that might be useful in light of the over-smoothing issue. Our proposed method performs competitively on citation datasets compared with other baseline methods.

- We make adaptions of the concept *identifiability* from [1] to GNNs, which originally refers to Transformers [23]. Generally speaking, preserving feature identity can boost performance of many computer vision and natural language learning tasks . However, our experiment result reveals that although a number of nodes maintain their identity across GNN layers, *node embedding identifiability* is not relevant to the accuracy of node classification problem.

- Based on an experiment of extremely deep *Graph Convolutional Network* (GCN) [3], we find that the classification result relies only on the graph structure when piling up enough layers. Together with the fact that *Multi-Layer Perception* (MLP) is a purely attribute-based model that does not consider the graph structure, we introduce the new concept *Graph Neural Network Necessity* (GNN-N), since GNNs take advantage of both node features and graph structures and are only necessary when other effortless alternatives cannot solve the problem properly. We then utilize GNN-N to appraise several most frequently used node classification datasets.

Chapter 2

Related Works

Fixed Point. Increasing depth and non-linearity can enhance representation power of many neural network architectures [24, 25, 26]. Motivated by an observation that the hidden layers of many existing deep sequence models converge towards some fixed point, several works has proposed different methods finding the fixed point to implicitly increase depth of networks. Deep Equilibrium Models (DEQ) [27] and Implicit Deep Learning (IDL) [28] find the fixed point by solving a system of equations, while Equilibrated Recurrent Neural Network (ERNN) [29] relies on augmenting the model with a time-delayed self-feedback loop. However, they are approaches for Recurrent Neural Networks (RNNs) not for GNNs. A more related work for GNNs is Steady-States Embedding (SSE) [30] that designs a learning method which alternates between updating the embeddings and projecting them onto the steady-state constraints to find fixed point. Compared with SSE, our proposed method by adding an additional loss function to find fixed point is more straightforward and run-time efficient.

Several recent works discuss the performance degradation when stacking many GNN layers. Since finding fixed points of node representations equals stacking infinite GNN layers, the performances should share some similarities. Research [31, 22] supports the explanation that the over-smoothing issue accounts for the performance degradation, which means repeatedly stacking layers may make representations of nodes converge to a same value or be proportional to the square root of the node degree. Approaches [32, 33, 34, 35, 36, 37] are proposed to alleviate over-smoothing issue. Several of them [33, 36, 37] reform from the topological view by modifying the graph topology. Xu *et al.* [32] leverage different neighborhood ranges to enable better structure-aware representation. Chen *et al.* [34] incorporate the initial node representation during feature propagation and add an identity matrix to the weight matrix during feature transformation to obtain robust representation.

Another prominent factor that compromises performances of deeper GNNs is the entanglement of representation transformation and propagation [38], which results in redundant parameters and unnecessary complexity. Reducing non-linearity [39] or representation transformation [38] can effectively improve the performance of deeper GNNs.

The above mentioned works stick to the traditional approach stacking GNN layers deeply, whereas we make an attempt to find fixed points directly after a very early GNN layer, which has not been affected by the over-smoothing issue yet. If the joint loss function in our approach is optimized to zero, the last hidden layer reaches a fixed point and thus can be viewed as an implicit layer in a broad sense. This apparently reduces the complexity and the number of parameters in the model compared to stacking infinite GNN layer.

Identifiability. Previous research has verified the importance of *identity* in deep learning [40]. In the field of computer vision, the standard parameterization makes it non-trivial for a convolutional layer trained with stochastic gradient methods, such as AlexNet [41], to preserve features that were already good. In another word, such convolutional layers can neither converge to the identity transformation nor retain feature identity. This shortcoming was observed and addressed by [42, 43] through *residual networks*, which explicitly introduced a reparameterization of the convolutional layers such that when all trainable weights are 0, the layer represents the identity function. Since then, residual networks and subsequent architectures have consistently achieved state-of-the-art results on various computer vision benchmarks such as CIFAR10 and ImageNet [43].

Regarding natural language processing, Brunner *et al.* evolve the concept *feature identity* to *token identifiability* [1], which is defined as the fine-grained, word-level mappings between input and output generated by a model [23]. They devise an experimental setting recovering input tokens from word embeddings with linear or non-linear perceptron and then finding the match in a nearest neighbour sense within the same input sentence. Their result gives substance to the hypothesis that contextual word embeddings maintain their identity as they pass through successive layers of a Transformer. A large number of further research is conducted based on this assumption [44, 45, 46, 47, 48, 49]. For instance, [48, 49] use classifiers to probe hidden embeddings for word-specific aspects without factoring in how much the word is still represented and [46, 47] sum the attention to a specific sequence position over layers and attention heads, while the given position might encode a different mixture of inputs in each layer.

Our work expands the above discussions about *identity* issue to the field of graph related problems. More specifically, we make adaptations on the concept *token identifiability* to graph-type data structure and introduce *node embedding identifiability*, which investigates the existence of mappings between input features of one node and its embeddings after GNN layers.

GNN-N. Chapelle *et al.* summarize conventional algorithms and models dealing *Semi-Supervised Learning* problems in the book [50]. The most representative algorithms that only consider graph structure and ignore node attributes are *Label Propagation*(LabelProp) and *Normalized Laplacian Label Propagation* (LabelProp NL). Both of them rely on the idea of building a graph whose nodes are data points (labeled and unlabeled) and edges represent similarities between points. LabelProp iteratively propagate known labels through neighbourhoods in order to label all nodes, while LabelProp NL takes an additional regularization term based on the graph Laplacian into account. In order to estimate the dependency of node classification problem on graph structures, we similarly need to neglect the impact of node attributes. We use extremely deep GCNs based on the fact that representations of nodes converge to a stationary distribution and the classification results rely no more on the input node features as depth increases to a certain extent (Appendix C). Known labels play a crucial part in LabelProp and LabelProp NL algorithms, whereas our method does not involve known labels in the inference and thus is more close to the way how normal GNNs utilize graph structure.

As mentioned in the former paragraph, the performance degradation when stacking many GNN layers has been observed and discussed in [31, 22, 32, 33, 38, 34, 35, 36, 37]. However, they lay the emphasis on finding reasons and overcoming this phenomenon, while we take advantage of it to evaluate the importance of graph structures for node classification problems. Along with the fact that MLP only considers node attributes, hence can be used to evaluate the importance of node attributes for node classification problems, we present a new concept *Graph Neural Network Necessity* that measures the relevance of a dataset with GNNs, i.e. to what extent the classification problem on this dataset should be solved with GNNs. To the best of our knowledge, we are the first to design a metric to assess the relevance of a graph related dataset with GNNs.

Chapter 3

Fixed Point

Numerous theoretical and empirical studies reveal that deep and non-linear architectures can enhance representation power of deep neural networks [24, 25, 26]. As depth increases, hidden layers of many existing deep sequence models converge towards some fixed point. Finding this fixed point directly is equivalent to running an infinite depth feedforward network, yet with much less memory consumption [27, 28, 29].

Likewise, fixed points play an important role in graph analytics problems. Many graph analytics problems can be solved via iterative algorithms where solutions are often characterized by reaching a fixed point, e.g. PageRank [51] or mean field inference.

Inspired by former works on deep sequence models, we propose a method by adding an additional loss function to minimize the difference between input and output of last hidden layer. If this additional loss function is optimized to zero, it implies that node representations reach some fixed point after last hidden layer. Assuming that the GNN model consists of N layers, $\mathbf{H}^n$ and $\mathbf{y}$ represent the node embeddings after n-th layer and the labels respectively. In such a model, N-th layer should be a classification layer mapping node embeddings to logits and $(N-1)$-th layer is the so-called last hidden layer. To construct the additional loss function to find the fixed points of $(N-1)$-th layer, we replicate it and concatenate the replica after the original $(N-1)$-th layer, where we note this replica layer as $N^{'}$-th layer to distinguish it from the original N-th layer. Fig.3.1 shows an example structure when $N = 3$. We use the mean squared error (MSE) between input and output of $N^{'}$-th layer as the additional loss function, i.e. $\|\mathbf{H}^{N^{'}} - \mathbf{H}^{N-1}\|^2$. The normal classification function should be a negative log-likelihood (NLL), i.e. $-\log Softmax(\mathbf{y}, \mathbf{H}^N)$. To construct a meaningful joint loss function, we then follow the approach proposed by Kendall *et al.* [52] to combine MLE with NLL.

3.1 Background

As the two separate loss functions may have different units and scales, they need to be combined in a reasonable way, so that GNNs can learn from them simultaneously and effectively. The most straightforward approach would be to simply perform a weighted linear sum of the two losses:

$$\mathcal{L}_{total}(\mathbf{W}) = \omega_1 \mathcal{L}_1(\mathbf{W}) + \omega_2 \mathcal{L}_2(\mathbf{W}), \tag{3.1}$$

where $\mathbf{W}$ represents model parameters. However, it's expensive to manually tune these weight hyper-parameters. To address this issue, Kendall *et al.* [52] propose a principled way weighting individual task's loss with homoscedastic task uncertainty.

Assume that a model's multiple outputs are composed of a continuous output $\mathbf{y}_1$ for regression and a discrete output $\mathbf{y}_2$ for classification, then the joint loss $\mathcal{L}(\mathbf{W}, \sigma_1, \sigma_2)$ is given as:

$$\mathcal{L}(\mathbf{W}, \sigma_1, \sigma_2) \approx \frac{1}{2{\sigma_1}^2}\mathcal{L}_1(\mathbf{W}) + \frac{1}{2{\sigma_2}^2}\mathcal{L}_2(\mathbf{W}) + \log \sigma_1 \sigma_2, \tag{3.2}$$

where we define $\mathcal{L}_1(\mathbf{W}) = \left\| \mathbf{y}_1 - \mathbf{f}^{\mathbf{W}}(\mathbf{x}) \right\|^2$ as the Euclidean loss of $\mathbf{y}_1$, define $\mathcal{L}_2(\mathbf{W}) = -\log Softmax(\mathbf{y}_2, \mathbf{f}^{\mathbf{W}}(\mathbf{x}))$ as the cross entropy loss of $\mathbf{y}_2$, and optimise with respect to $\mathbf{W}$ as well as σ_1, σ_2. Weights are discouraged from increasing too much by the last term in the joint loss, which acts as a regulariser.

3.2 Setup of the Experiment

In this experiment, we train GNNs with the joint loss, which consists of a cross entropy loss for classification and an additional mean squared error described formerly, to investigate whether our attempt to find fixed points can improve the performance.

Among all variants of GNNs, Graph Convolutional Networks (GCNs) is one of the most representative methods. Thus, we choose to train GCNs on three established citation datasets [53], namely Cora, PubMed and CiteSeer, to demonstrate the result. The task is to do node classification and we use the same training/validation/test splits and follow other experimental setup closely as in [3], except using 3-layers instead of 2-layer GCN, since we need at least one hidden layer which has equal input and output feature size in order to compute the additional loss function. Fig.3.1 shows the concrete structure of the 3-layers GCN used in the experiment.

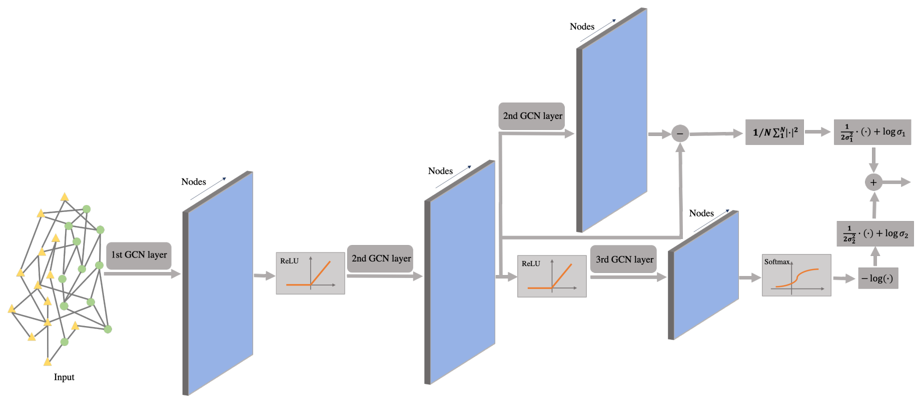


Figure 3.1: Structure of 3-layer GCN model used in the experiment.

3.3 Experiment Results

Results are summarized in Tab.3.1 and Fig.3.2. Reported numbers denote classification accuracy in percent. Results of Graph Convolutional Networks (GCNs) and Steady-State Embedding (SSE) are taken directly from [3, 30] respectively and serve as baselines.

Our proposed method performs competitively on all of the three citation datasets. Because the number of random seeds used in experiments was limited, we cannot exclude the possibility that the slight improvement on PubMed is by chance. Yet, our proposed method has advantages like reduced complexity, much less parameters and time-efficient compared to SSE.

Dataset	GCN	SSE	GCN with additional Loss
Cora	**81.5**	79.4	81.2
PubMed	79.0	75.8	**79.7**
CiteSeer	70.3	**72.5**	69.0

Table 3.1: Results in table format.

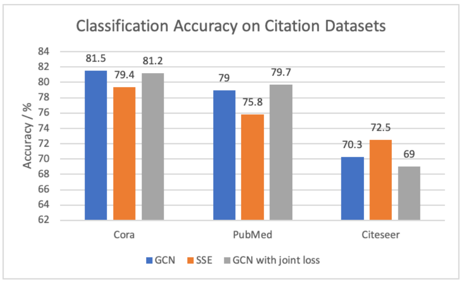


Figure 3.2: Results in figure format.

Chapter 4

Identifiability

Identity matters in Deep Learning [40]. In the field of computer vision, residual networks and subsequent architectures have consistently achieved state-of-the-art results on various benchmarks [42, 43]. One important reason lies in their ability to preserve the identity of features. Meanwhile, in the field of natural language processing, Brunner *et al.* [1] justified with plenty of evidence that contextual word embeddings maintain their identity as they pass through successive layers of a transformer, which is arguably a desirable property, as it affects the replicability and interpretability of the model's predictions.

This naturally raises questions for GNNs, whether node embeddings maintain their identity as well and what impact it has on the performance. To investigate those questions, we borrow the concept *token identifiability* from [1] and introduce a similar one - *node embedding identifiability* for GNNs.

4.1 Background

Brunner *et al.* [1] introduce the concept of *token identifiability* as the existence of a mapping assigning contextual embedding to their corresponding input tokens, which can be adapted to node embeddings.

A node embedding $\mathbf{h}_i^l$ is identifiable if there exists a classification function $c(\cdot)$ such that $c(\mathbf{h}_i^l) = \mathbf{x}_i$, where $\mathbf{x}_i$ is the input node feature. Since the graph structure might be complex, our attempts show that recovering and identifying node features in the range of whole graph is difficult. Hence, we only require $c(\mathbf{h}_i^l)$ to recover $\mathbf{x}_i$ in a nearest neighbour sense and find the match within one-hop neighbourhoods, i.e. we define $c_l(\cdot) = NN(g_l(\cdot))$ for each layer l , where $NN(\cdot)$ is the nearest neighbour lookup within one-hop neighbourhoods and $g_l : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$ is a continuous function mapping node embeddings $\mathbf{h}_i^l$ to vectors of real numbers. In the experiment, g_l is approximated by MLP $\hat{g}_l^{MLP}(\mathbf{h}_i^l) = \hat{\mathbf{x}}_i$, trained on a dataset of $(\mathbf{h}_i^l, \mathbf{x}_i)$ pairs.

4.2 Setup of the Experiment

We use GCN [3] and Cora dataset [53] to visualize the experiment results in this section. In Appendix B.1 we provide results on two additional datasets.

Firstly, we train GCN models with different depth varying from 1-layer to 10-layer on Cora dataset. Feature size of all intermediate hidden layers is set to 16. Other hyper parameters for training GCN models are set following [3]. Then, we extract input features $\mathbf{x}_i$ and hidden node embeddings $\mathbf{e}_i^l$ from each layer of GCN models and train a 3-layer MLP $\hat{g}_l^{MLP}$ on extracted $(\mathbf{h}_i^l, \mathbf{x}_i)$ pairs using cosine distance as loss function for layer l. The recovered vectors $\hat{\mathbf{x}}_i$ are subsequently used to find the nearest neighbourhood within input features of node i 's one-hop neighbourhoods. Fig.4.1 shows the example setting for 3-layer GCN model.

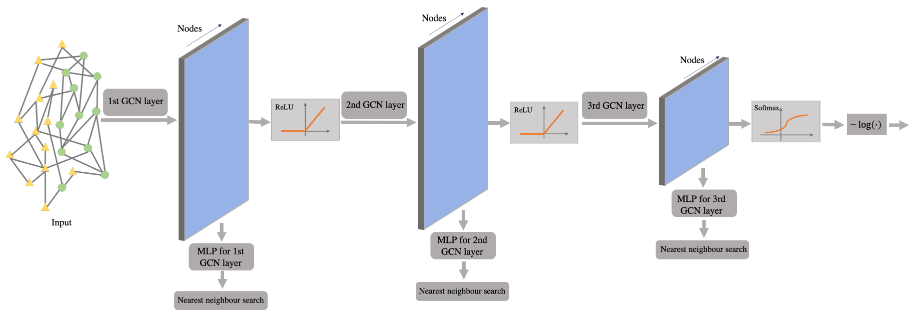
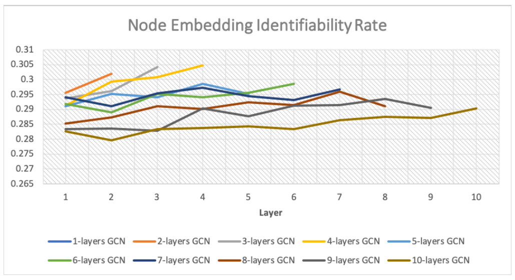


Figure 4.1: Example experiment setting when using 3-layer GCN model to integrate the identifiaility of nodes.

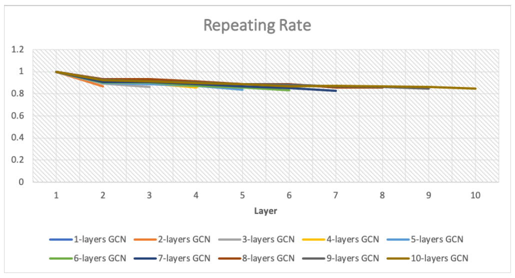
4.3 Experiment Results

In this section we report experiment results on Cora dataset. Node embedding identifiability defined as the percentage of correctly identified nodes is shown in Fig.4.2(a). To investigate whether nodes maintain their identifiability across layers, we compute the overlap proportion between identifiable nodes after 1st layer and identifiable nodes after n-th layer. We name this value as *repeating rate* for n-th layer and present it in Fig.4.2(b). Furthermore, we also report the classification accuracy of different layers' GCN models in Fig.4.2(c) and accuracy on identifiable and non-identifiable nodes respectively in Fig.4.2(d). If node embedding identifiability has impact on the performance, we should expect different accuracy on identifiable and unidentifiable nodes.

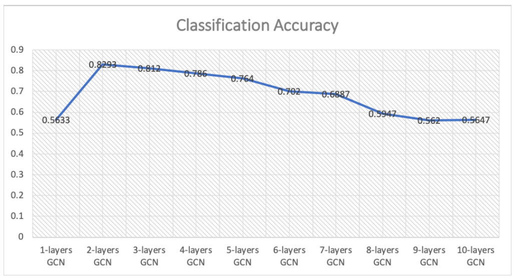
As can be seen from the result, node embedding identifiability rate fluctuates around 30% and repeating rate remains very high for every layer, which means about one third of nodes can be identified and most of them maintain their identifibility across layers. As the accuracy on identifiable nodes and unidentifiable nodes almost coincide with each other, we can draw the conclusion that the performance of GNNs doesn't rely on node embedding identifiability.



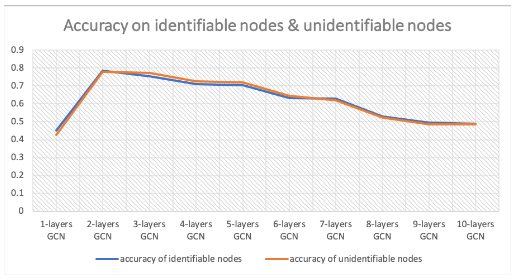
(a) node embedding idetifiability rate



(b) repeating rate



(c) classification accuracy



(d) accuracy on identifiable nodes and unidentifiable nodes

Figure 4.2: Experiment results on Cora.

Chapter 5

Graph Neural Network Necessity

GNNs suffer from performance degradation as piling up many layers and adding non-linearity. The reason lies in the over-smoothing issue [31, 22, 32, 33]. Node representations will converge to the same value or be proportional to the square root of the node degree, which means that the classification results rely more and more on graph structure when GNN models become deeper.

We observe that for 100-layer GCN model, classification results are no more related to input features on most datasets, i.e. original input features or any random features lead to same results (see Appendix C). However, the performance of 100-layer GCN can vary enormously on different datasets, revealing that prediction on different datasets depends on graph structure to a varying extent. Similar conclusion can be drawn for node features based on the performance of MLPs, which rely on node features and neglect the impact of graph structures.

GNNs utilize both node features and graph structure to make inference. Provided that the problem can be solved purely with graph structure or node features, GNNs are not necessary. From this perspective, we introduce a metric *Graph Neural Network Necessity* to measure the relevance of a dataset to GNNs based on the dataset's dependency on graph structures and node features.

5.1 Methodology

We focus on the node classification problem and define *Graph Neural Network Necessity* as the average possibility that a node from one specific dataset needs to be solved with GNNs, i.e. cannot be solved exclusively with graph structure or node features. Because only then, GNNs show their advantages by using information from both graph structure and node features and thus are necessary.

In mathematical expressions:

$$GNN\text{-}N = \frac{1}{N}\sum_{i=1}^{N} P_{i,GNNs} \tag{5.1}$$

$$= \frac{1}{N}\sum_{i=1}^{N}(1 - P_{i,graph\,structure})(1 - P_{i,features}), \tag{5.2}$$

where $P_{i,GNNs}$ represents the possibility that GNNs are necessary for node i. On the other hand, $P_{i,graph\,structure}$ and $P_{i,features}$ represent possibilities that node i can be correctly classified with graph structure or node features alone. Eq.5.2 uses the second definition of GNN-N and assumes $P_{i,graph\,structure}$ and $P_{i,features}$ are uncorrelated with each other (see Fig.5.1 Left).

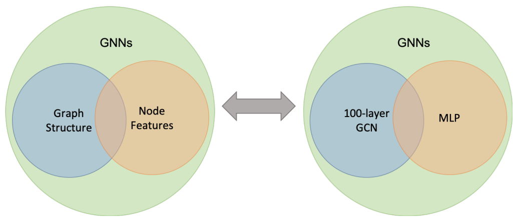


Figure 5.1: Left: Green, blue and orange circles represent $P_{i,GNNs}$, $P_{i,graph\,structure}$ and $P_{i,features}$ respectively. Node i can be solved purely with graph structure or node features. When neither of them work, GNNs are necessary. Right: We replace $P_{i,graph\,structure}$ and $P_{i,features}$ with experimental possibilities $\hat{P}^{MLP}_{i,features}$ and $\hat{P}^{100\text{-}layer\,GCN}_{i,graph\,structure}$ based on the fact that MLP relies only on node features while 100-layer GCN, on the contrary, relies only on graph structure.

Now the problem becomes how to obtain $P_{i,graph\,structure}$ and $P_{i,features}$. We use $\hat{P}^{MLP}_{i,features}$, the experimental possibility that node i can be correctly classified with MLP, to approximate $P_{i,features}$, based on the fact that MLP only considers node features.

For graph structure, we exploit the over-smoothing issue of GNNs: sufficiently deep GNNs no longer contain feature information. This phenomenon has been discovered in previous research but none of them mentioned how deep one GNN model should be to reach convergence point. We verify their conclusion via an experiment and prove that node presentations after 100-layer GCN have converged to a static distribution on most datasets. Namely, the classification results of 100-layer GCN with random node features are exactly the same as with original node features, which means the results are no more influenced by node features and only depend on the graph structure. More details are provided in Appendix C. On the premise of this experiment, we choose 100-layer GCN to compute the dependency of a dataset on graph structure. More precisely, we replace $P_{i,graph\,structure}$ in Eq.5.2 with $\hat{P}^{100\text{-}layer\,GCN}_{i,graph\,structure}$ representing the experimental possibility that node i can be correctly classified with 100-layer GCN.

Hence, to compute GNN-N of a dataset, we simply need to experiment with MLP and 100-layer GCN and compute the experimental possibilities $\hat{P}^{MLP}_{i,features}$ and $\hat{P}^{100\text{-}layer\,GCN}_{i,graph\,structure}$ (see Fig.5.1 Right).

5.2 Setup of the Experiment

We evaluate 7 node classification datasets based on GNN-N defined above. The details and statistics of these datasets are provided in Appendix A. To compute $\hat{P}^{MLP}_{i,features}$ and $\hat{P}^{100\text{-}layer\,GCN}_{i,graph\,structure}$, 3-layer MLP and 100-layer GCN are used in the experiment. The feature size of all GCN and MLP's intermediate hidden layers is set to 16. We train and test MLP on each dataset 10 times. For 100-layer GCN, we train them once and then test with 10 different random features and repeat this process 10 times on each dataset. After that, we count for each node how many times it's correctly classified and compute the experimental possibility $\hat{P}^{MLP}_{i,features}$ and $\hat{P}^{100\text{-}layer\,GCN}_{i,graph\,structure}$.

5.3 Experiment Results

The obtained GNN-N values of the 7 node classification datasets are presented in this section. Meanwhile, some auxiliary results, like the test accuracy of MLP and GCN, are provided in Appendix B.2 and Appendix C.

According to the obtained GNN-N values (see Fig.5.2), *CiteSeer* is most relevant to GNNs and its GNN-N is about 0.41 followed by *Cora* and *PubMed* with GNN-N values above 0.25. Conversely, almost all nodes in *Coauthor physics* can be correctly classified without GNNs and hence GNN are not so necessary for it. GNN-N values of other datasets vary from about 0.1 to about 0.26.

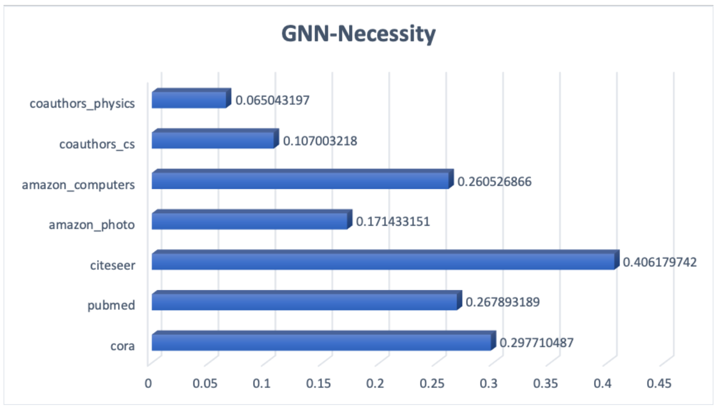


Figure 5.2:

Chapter 6

Conclusions

We attempt to gain a better understanding of GNNs' working principles from three different angles. Firstly, we propose a method to find fixed points of node embeddings by combining a MSE that minimizes the difference between input and output of last hidden layer and a NLL for the graph related classification task into a reasonable joint loss. Our method achieves competitive performance as baseline methods on three citation datasets yet with less time-consumption and reduced complexity. Additionally, we introduce the discussion about *identity* from other fields of deep learning to GNNs. Our experiment reveal that a number of nodes retain their identity across GNN models, but *node embedding identifiability* has no impact on performance of GNNs. Lastly, we propose a metric *graph neural network necessity* for graph related dataset to measure their relevance to GNNs. We evaluate several datasets with GNN-N and find citation datasets are generally more related to GNNs compared to co-purchase and coauthor datasets.

There remain some interesting aspects that deserve to be studied further. For the sake of simplicity, we made the assumption that $P_{i,graph\,structure}$ and $P_{i,features}$ are uncorrelated to derive Eq.5.2. Taking more complex and realistic relationships between $P_{i,graph\,structure}$ and $P_{i,features}$ into consideration can be one of the future research directions. Besides, we mainly focus on node classification problems in this paper and the discussion should be generalized to other graph related tasks, such as graph classification and link prediction in the future work.

Bibliography

[1] G. Brunner, Y. Liu, D. Pascual, O. Richter, M. Ciaramita, and R. Wattenhofer, “On identifiability in transformers,” 2019.

[2] W. L. Hamilton, R. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” 2017.

[3] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” 2016.

[4] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, “Graph attention networks,” 2017.

[5] P. Yanardag and S. Vishwanathan, “Deep graph kernels,” in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015, p. 1365–1374.

[6] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. V. N. Vishwanathan, A. J. Smola, and H.-P. Kriegel, “Protein function prediction via graph kernels,” *Bioinformatics*, vol. 21, no. 1, p. 47–56, Jan. 2005.

[7] D. P. Dobson and J. A. Doig, “Distinguishing enzyme structures from non-enzymes without alignments,” *Journal of Molecular Biology*, pp. 771–783, 2003.

[8] I. Schomburg, A. Chang, C. Ebeling, M. Gremse, C. Heldt, G. Huhn, and D. Schomburg, “Brenda, the enzyme database: Updates and major new developments,” *Nucleic acids research*, vol. 32, pp. D431–3, 01 2004.

[9] M. Togninalli, E. Ghisu, F. Llinares-López, B. Rieck, and K. Borgwardt, “Wasserstein weisfeiler-lehman graph kernels,” 2019.

[10] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, “Convolutional networks on graphs for learning molecular fingerprints,” 2015.

[11] H. Dai, B. Dai, and L. Song, “Discriminative embeddings of latent variable models for structured data,” 2016.

[12] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, “Neural message passing for quantum chemistry,” 2017.

[13] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, “How powerful are graph neural networks?” 2018.

[14] N. Peng, H. Poon, C. Quirk, K. Toutanova, and W. tau Yih, “Cross-sentence n-ary relation extraction with graph lstms,” 2017.

[15] H. Gao, Z. Wang, and S. Ji, “Large-scale learnable graph convolutional networks,” *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Jul 2018.

[16] Y. Ma, S. Wang, C. C. Aggarwal, and J. Tang, “Graph convolutional networks with eigenpooling,” 2019.

[17] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, “An end-to-end deep learning architecture for graph classification,” *AAAI*, 2018.

[18] L. Cai and S. Ji, “A multi-scale approach for graph link prediction,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 3308–3315, 04 2020.

[19] M. Zhang and Y. Chen, “Weisfeiler-lehman neural machine for link prediction,” in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM, aug 2017.

[20] M. Zhang and Y. Chen, “Link prediction based on graph neural networks,” 2018.

[21] R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, “Hierarchical graph representation learning with differentiable pooling,” 2018.

[22] Q. Li, Z. Han, and X.-M. Wu, “Deeper insights into graph convolutional networks for semi-supervised learning,” 2018.

[23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” 2017.

[24] M. Telgarsky, “Benefits of depth in neural networks,” 2016.

[25] M. Chen, J. Pennington, and S. S. Schoenholz, “Dynamical isometry and a mean field theory of rnns: Gating enables signal propagation in recurrent neural networks,” 2018.

[26] P. Zhou and J. Feng, “Understanding generalization and optimization performance of deep cnns,” 2018.

[27] S. Bai, J. Z. Kolter, and V. Koltun, “Deep equilibrium models,” 2019.

[28] L. E. Ghaoui, F. Gu, B. Travacca, and A. Askari, “Implicit deep learning,” 2019.

[29] Z. Zhang, A. Kag, A. Sullivan, and V. Saligrama, “Equilibrated recurrent neural network: Neuronal time-delayed self-feedback improves accuracy and stability,” 2019.

[30] H. Dai, Z. Kozareva, B. Dai, A. Smola, and L. Song, “Learning steady-states of iterative algorithms over graphs,” in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 80, 10–15 Jul 2018, pp. 1106–1114.

[31] K. Oono and T. Suzuki, “Graph neural networks exponentially lose expressive power for node classification,” 2019.

[32] K. Xu, C. Li, Y. Tian, T. Sonobe, K. ichi Kawarabayashi, and S. Jegelka, “Representation learning on graphs with jumping knowledge networks,” 2018.

[33] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, “Measuring and relieving the over-smoothing problem for graph neural networks from the topological view,” 2019.

[34] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, “Simple and deep graph convolutional networks,” 2020.

[35] A. Hasanzadeh, E. Hajiramezanali, S. Boluki, M. Zhou, N. Duffield, K. Narayanan, and X. Qian, “Bayesian graph neural networks with adaptive connection sampling,” 2020.

[36] C. Zheng, B. Zong, W. Cheng, D. Song, J. Ni, W. Yu, H. Chen, and W. Wang, “Robust graph representation learning via neural sparsification,” in *International Conference on Machine Learning*, 2020.

[37] Y. Rong, W. Huang, T. Xu, and J. Huang, “Dropedge: Towards deep graph convolutional networks on node classification,” 2019.

[38] M. Liu, H. Gao, and S. Ji, “Towards deeper graph neural networks,” 2020.

[39] F. Wu, T. Zhang, A. H. de Souza Jr., C. Fifty, T. Yu, and K. Q. Weinberger, “Simplifying graph convolutional networks,” 2019.

[40] M. Hardt and T. Ma, “Identity matters in deep learning,” 2016.

[41] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS’12, 2012, p. 1097–1105.

[42] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” 2015.

[43] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," 2016.

[44] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2018.

[45] J. Vig, "Visualizing attention in transformer-based language representation models," 2019.

[46] K. Clark, U. Khandelwal, O. Levy, and C. D. Manning, "What does bert look at? an analysis of bert's attention," 2019.

[47] J. Baan, M. ter Hoeve, M. van der Wees, A. Schuth, and M. de Rijke, "Do transformer attention heads provide transparency in abstractive summarization?" 2019.

[48] Y. Lin, Y. C. Tan, and R. Frank, "Open sesame: Getting inside bert's linguistic knowledge," 2019.

[49] M. E. Peters, M. Neumann, L. Zettlemoyer, and W. tau Yih, "Dissecting contextual word embeddings: Architecture and representation," 2018.

[50] O. Chapelle, B. Schlkopf, and A. Zien, *Semi-Supervised Learning.* The MIT Press, 2010.

[51] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Technical Report 1999-66, November 1999.

[52] A. Kendall, Y. Gal, and R. Cipolla, "Multi-task learning using uncertainty to weigh losses for scene geometry and semantics," 2017.

[53] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad, "Collective classification in network data," *AI Magazine*, vol. 29, pp. 93–106, 2008.

[54] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," 2018.

[55] J. McAuley, C. Targett, Q. Shi, and A. van den Hengel, "Image-based recommendations on styles and substitutes," 2015.

Appendix A

Description and Statistics of Datasets

Citation Datasets. We consider three citation datasets[53]: Cora, PubMed and CiteSeer. The task is to do document classification, where each node in the citation graph represents the corresponding document. The documents have auxiliary bag-of-words features. The number of features corresponds to the vocabulary size in each dataset. The undirected edges are formed by the citation relationship between articles. We use the same training/validation/test splits as in [3]. During training, only 20 instances per class are provided with corresponding labels.

Co-purchase Datasets. Amazon Computers and Amazon Photo [54] are segments of the Amazon co-purchase graph [55], where nodes are goods and edges denote that two goods are frequently bought together. Node features are derived from bag-of-words representations for product reviews and class labels are given by the product category. We utilize 20 labeled nodes per class as the training set, 30 nodes per class as the validation set, and the rest as the test set.

Co-authors Datasets. Coauthor CS and Coauthor Physics [54] are co-authorship graphs datasets. Nodes denote authors, which are connected by an edge if they co-authored a paper. Node features represent paper keywords for each author's papers. Each node has a label denoting the most active fields of study for the corresponding author. Likewise, only 20 labeled nodes are used for training.

More detailed statistics of datasets are listed below in Tab.A.1.

Dataset	**#Classes**	**#Nodes**	**#Edges**	**Edge Density**	**#Features**	**#Training Nodes**	**#Validation Nodes**	**#Test Nodes**
Cora	7	2708	5278	0.0014	1433	20 per class	500	1000
PubMed	3	19717	44324	0.0002	500	20 per class	500	1000
CiteSeer	6	3327	4552	0.0008	3703	20 per class	500	1000
Amazon Photo	8	7487	119043	0.0042	745	20 per class	30 per class	rest nodes
Amazon Computers	10	13381	245778	0.0027	767	20 per class	30 per class	rest nodes
Coauthors CS	15	18333	81894	0.0005	6805	20 per class	30 per class	rest nodes
Coauthors Physics	5	34493	247962	0.0004	8415	20 per class	30 per class	rest nodes

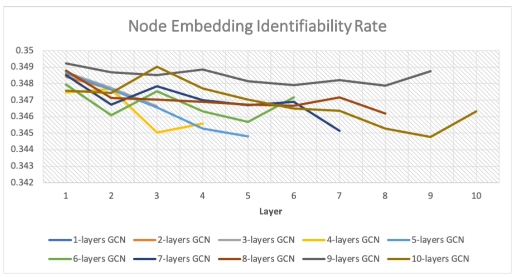
Table A.1: Statistics of datasets.

Appendix B

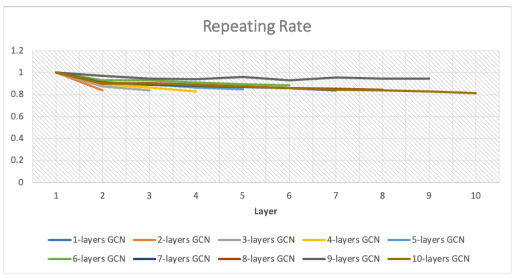
Additional Experiment Results

B.1 Identifiability

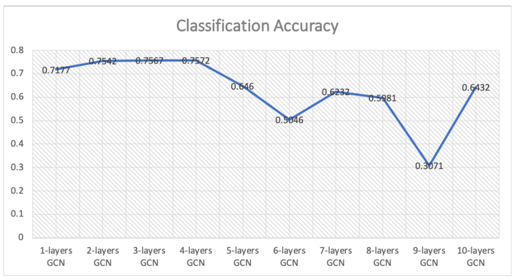
We provide experiment results on CiteSeer and PubMed in Fig.B.1 and Fig.B.2. Similar conclusions can be drown as in Chapter 4. About one third of nodes from PubMed or CiteSeer are identifiable and the high *repeating rate* values imply that those identifiable nodes retain their identifiability across layers. However, accuracy on identifiable nodes and unidentifiable nodes coincide with each other, which means accuracy is not influenced by *node embedding identifiability*.



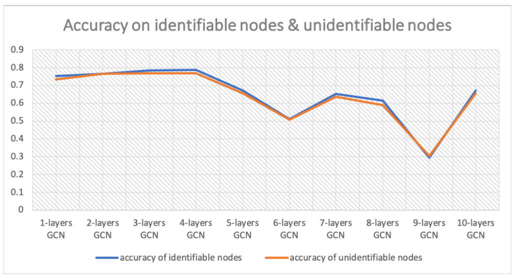
(a) node embedding identifiability rate



(b) repeating rate

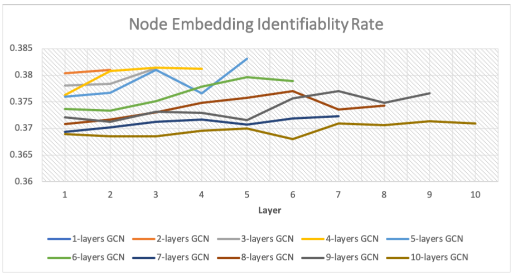


(c) classification accuracy

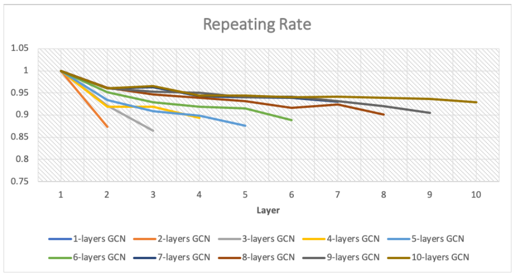


(d) accuracy on identifiable nodes and unidentifiable nodes

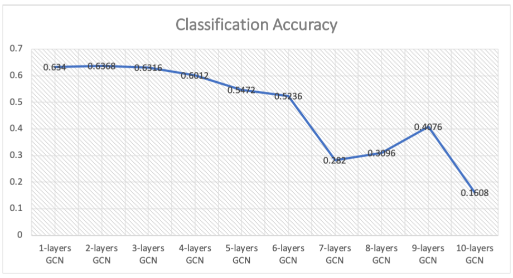
Figure B.1: Experiment results on PubMed dataset



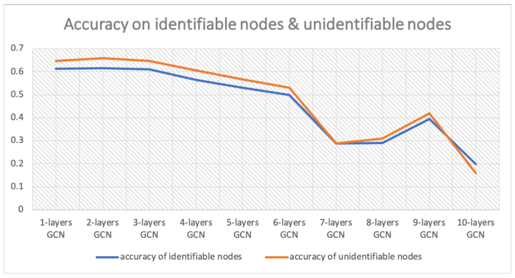
(a) node embedding identifiability rate



(b) repeating rate



(c) classification accuracy



(d) accuracy on identifiable nodes and unidentifiable nodes

Figure B.2: Experiment results on CiteSeer dataset

B.2 GNN-Necessity

We present auxiliary results during computation of GNN-N values for each dataset here. Results related to node features, i.e. experiment results with MLP, are visulized and summerized in this section. When computing dependency on graph structure, we directly use the results from the experiment of 100-layer GCN described in Appendix C. To avoid repetition, they are not included in this section.

The test accuracy of MLP on each dataset is presented in percent. Meanwhile we compute the *repeating rate* defined as Eq.B.1

$$repeating\,rate = \frac{|set_1 \cap set_2|}{\min\{|set_1|, |set_2|\}}, \tag{B.1}$$

where set_1 and set_2 represent correctly classified nodes from two different experiment trials, $|\,\boldsymbol{.}\,|$ represent the operation calculating the number of elements inside one set. Results are shown in Fig.B.3.

The test accuracy and repeating rate of the two coauthor datasets are higher than other datasets, which means they depend highly on node features to classify nodes. On the other hand, classification on citation datasets is not so closely bonded to node features.

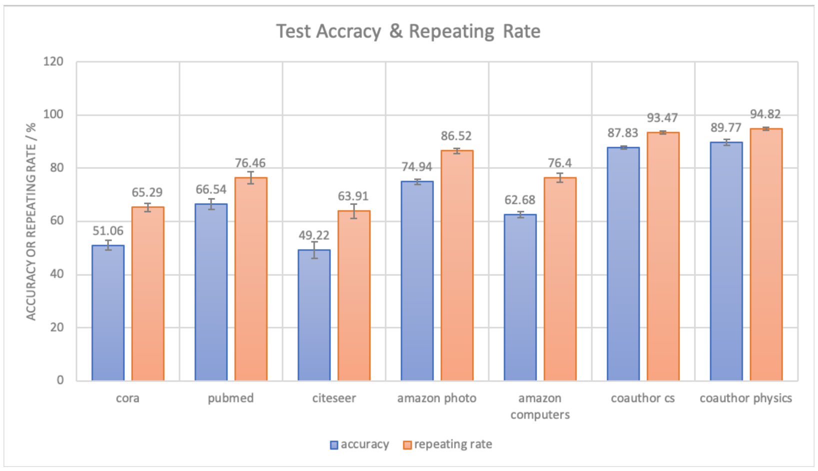


Figure B.3: Test accuracy and repeating rate on each dataset with standard variance.

Appendix C

The Experiment of 100-layer GCN

In this section we describe the experiment and results with 100-layer GCN, which is a premise for the computation of GNN-N value. We train 100-layer GCN 10 times and for each training trial we test the trained model with 10 different random features as well as with original features. Then we compare the correctly classified nodes from different tests. The comparison results are processed and presented via 3 different *repeating rates*, namely *random features-repeating rate* (R-RR), *random versus original features-repeating rate* (RO-RR) and *training trials-repeating rate* (TT-RR).

All of the above mentioned repeating rates can be defined by Eq.B.1, yet with different interpretations of set_1 and set_2 (see Tab.C.1) [1]. To note is that we only use the first random feature to compute RO-RR and TT-RR, since we already know the result that R-RR of all datasets are 1 and the first random feature can thus represent all the others.

Repeating Rate	set_1	set_2
R-RR	arbitrary random feature $\mathbf{r}$ used in training trial $\mathbf{t}$	another random feature $\mathbf{r}'$ used in training trial $\mathbf{t}$
RO-RR	the first random feature used in training trial $\mathbf{t}$	original feature
TT-RR	the first random feature used in training trial $\mathbf{t}$	the first random feature used in another training trial $\mathbf{t}'$

Table C.1: Explanation of different repeating rates.

In another word, repeating rates reflect whether it's always the same set of nodes that are correctly classified under different test situations (see Fig.C.1). R-RR compares between two arbitrary random features, while RO-RR compares

[1]We describe the input node features and the training trial of tests in Tab.C.1, however set_1 and set_2 represent the correctly classified nodes from the specific test.

between random features and original ones. If R-RR and RO-RR are both close to 1, it means input node features have little impact on classification results, i.e. the classification results rely mainly on graph structures. TT-RR compares between models from different training trials. High TT-RR implies that a number of nodes can be solved purely with graph structures, whereas low TT-RR indicates nodes that are correctly classified purely with graph structures are random.

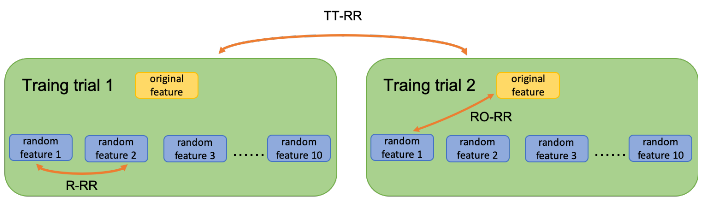


Figure C.1: Explanation of different repeating rates.

Repeating rates and test accuracy are shown in Fig.C.2 and Fig.C.3. The accuracy is computed separately for random features and original features. R-RR and RO-RR of all datasets are close to 1, which suggests that node presentations after 100-layer GCN have converged to a static distribution and classification results rely no more on input node features. TT-RR of Cora, CiteSeer, Amazon computers and Coauthor physics are high, hence some nodes inside these four datasets can be correctly classified purely with graph structure.

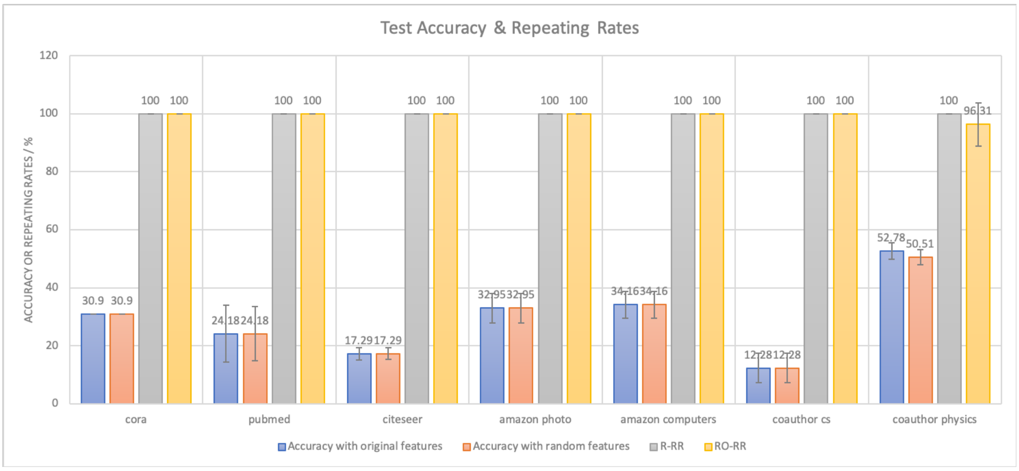
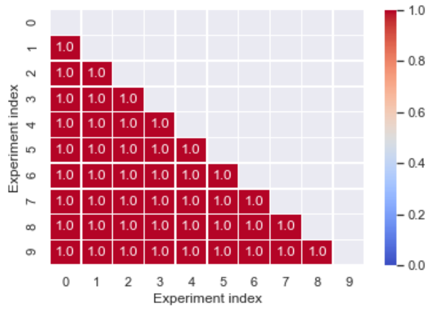
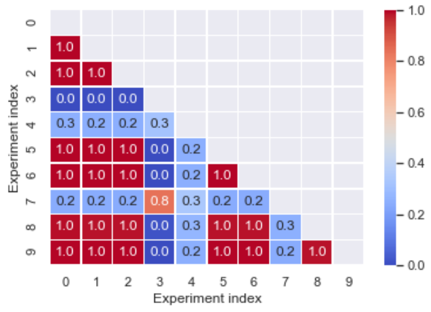


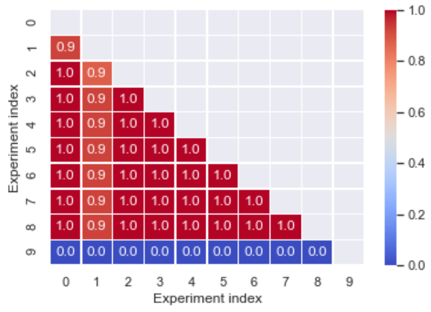
Figure C.2: Test accuracy with random/original features, R-RR and RO-RR of several node classification datasets with standard variance. R-RR and RO-RR of all datasets are close to 1.



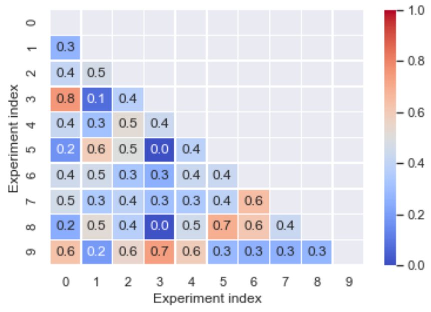
(a) Cora



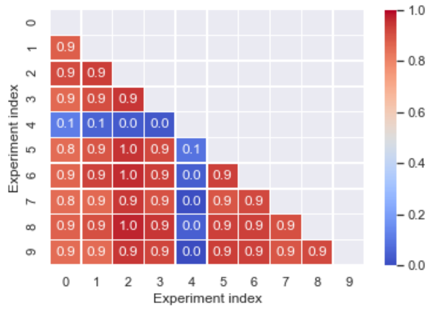
(b) PubMed



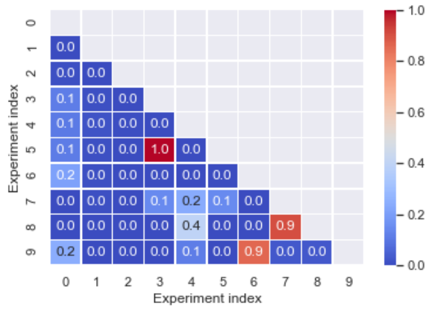
(c) CiteSeer



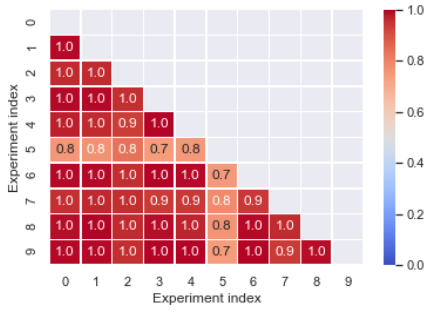
(d) Amazon photo



(e) Amazon computers



(f) Coauthor CS



(g) Coauthor physics

Figure C.3: TT-RR visualized with heatmaps. Cora, CiteSeer, AMZ computers and Coauthor physics have higher TT-RR, which indicates a number of nodes can be correctly classified purely with graph structures. On the contrary, classification results purely with graph structure are almost random on PubMed, AMZ photo and Coauthor CS.