A Survey of Graph Neural Networks on Synthetic Data

Brigham Stone Carson
Brigham Young University

A Survey of Graph Neural Networks on Synthetic Data

Brigham Stone Carson

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Mathematics

Brigham Young University

ABSTRACT

A Survey of Graph Neural Networks on Synthetic Data

Brigham Stone Carson
Department of Mathematics, BYU
Master of Science

We relate properties of attributed random graph models to the performance of GNN architectures. We identify regimes where GNNs outperform feedforward neural networks and non-attributed graph clustering methods. We compare GNN performance on our synthetic benchmark to performance on popular real-world datasets. We analyze the theoretical foundations for weak recovery in GNNs for popular one- and two-layer architectures. We obtain an explicit formula for the performance of a 1-layer GNN, and we obtain useful insights on how to proceed in the 2-layer case. Finally, we improve the bound for a notable result on the GNN size generalization problem by 1.

Keywords: graph neural networks, machine learning, network theory

Acknowledgements

Thank you first and foremost to my advisor, Zach Boyd: you have been such a great mentor; thank you for guiding me through the program, for offering useful career advice, and for the office hours where your insights pushed the project along.

Huge thank you also to Drake Brown, one of the brightest people I know and resident ML implementation whiz. Thanks for listening to me ramble about various topics in mathematics, and for being a great sounding board for research ideas. You're going places!

Thanks to the other occupants of TMCB 300, including but not limited to Ian, Mats, and Cole, for being down to chat about projects, grad student life, and other things. You guys make the lab an interesting place to be!

Thanks Jonathan for weekly fishing trips and great conversations about mental health. And, of course, thank you to my wife and family for all of the support it takes to keep a math grad student healthy and functioning. And thanks to me for not giving up.

The following Master's thesis is unofficially brought to you by the following: two pieces of peanut butter and jelly toast every morning, a jarring mix of Coldplay and Pearl Jam, the 70lb dumbbells at VASA, a bonefish, a caribou, that time I almost slipped off a 40ft cliff but didn't, and an alarming amount of Mountain Dew. Enjoy.

Contents

List of Figures

Chapter 1. Introduction

With the recent release of powerful models like GPT-3 and DALL-E 2, machine learning has exploded into the public eye, with more and more headlines about the future of artificial intelligence appearing each day. But how does it all work? The goal of most machine learning architectures is to estimate a statistical model that captures the semantic features of a training dataset and generalize them to broader, unseen data. Thus the more information about the data we can provide to such a model, the more characteristics the model can learn. Some common architectures include feedforward neural networks, which operate on isolated points of data; convolutional neural networks, which operate on images; and transformers which operate on sequential data like text strings.

In today's world, useful data often exhibits a richer structure than that of a collection of isolated points. The structure of data is not always regular in shape like an image, and there is not always a clear or useful ordering of the data for sequential models. Much of the data used in applications today is better represented as a network. Networks are loosely defined as collections of objects and the relationships between them. In practice, networks are usually represented as graphs with associated vertex and edge sets. Graph theory is a well-studied area of mathematics that has garnered increased interest from scholars and practitioners alike in the information age. The theory of networks and graphs has applications in nearly every scientific and industrial domain, from bioinformatics, to supply chain management, to the social sciences.

In this thesis we will combine approaches from both network theory and machine learning to better understand graph neural networks (GNNs), a class of machine learning models that leverage network structure to gain further information about the data. To do this we employ random graph generation techniques to control properties of the input data and gain a theoretical handle on GNN performance.

To summarize, this thesis documents the following contributions:

- A novel method of benchmarking GNN performance through usage of synthetic datasets
- A series of computational experiments validating observed GNN phenomena on synthetic datasets
- A novel approach to weak recovery analysis for a popular GNN architecture.

Chapter 2. Testing GNNs on Stochastic Block Models

2.1 Introduction

Graph neural networks (GNNs) have achieved impressive success on node classification tasks due to their ability to harness graph topology in the learning process [3]. Because many GNNs assume homophily among nodes in the graph, it is often believed that GNNs don't perform as well on heterophilous graphs [4]. While several architectures have been proposed to recover the performance of GNNs on such data [5], most models are tested on a limited selection of benchmark datasets, which fails to capture the broad range of structure possible in naturally occurring data. To design more effective GNNs, it is necessary that we understand how various properties of the data (e.g., modularity, degree distribution, or feature separability) affect model performance. By understanding such relationships, we can construct stronger architectures that more efficiently take advantage of the data's structural qualities.

To shed light on these relationships, we use random graph models to generate collections of training graphs with predetermined characteristics. Stochastic block models [2], already standard for evaluating many graph algorithms, offer a well documented random graph model which is suitable for node classification tasks when adjoined to feature information. Note when referring to data generation methods we use the term generative models while *model* will refer to a trained architecture.

In this chapter we:

- report the performance of popular GNN architectures over a spectrum of synthetic data;
- identify graph regimes where GNNs outperform graph-agnostic and feature-agnostic clustering methods;

- outline the degree to which edge and feature information contribute to overall performance in both homophilous and heterophilous datasets; and
- contextualize GNN performance on common benchmark datasets, such as citation networks, with synthetically generated data.

2.2 Previous Work

In 2022 there have been numerous papers exploring the theoretical bounds for various GNN architectures. [6] explored in which regimes the attention module in the Graph Attention Network (GAT) [7] contributes a meaningful difference in performance. Following this, [8] proved theoretically that utilizing edge data via graph convolutions expands the range in which a neural network can correctly classify nodes by $\frac{1}{\sqrt[4]{\text{avg. degree}}}$. Lastly, [9] found that a Graph Convolutional Network (GCN) only performs low pass filtering on the feature vectors.

In addition, a number of works have explored various methods of generating realistic graph data. Central to this work are those that use stochastic block models (SBM). [2] summarized the key mathematical findings related to SBMs and outlined their limitations. [10] extended the SBM to the *degree-corrected SBM*, which allows for heavy-tailed degree distributions. Degree-corrected SBMs have been subjected to heavy theoretical analysis in recent years. For example, [11] derived asymptotic minimax risks for misclassification in degree-corrected SBMs. Lastly, [12] proposed a *contextual SBM*(cSBM) that generates feature data alongside the graph data. This was originally proposed to analyze specific properties of belief propagation (BP) [13].

Additional work has been done studying the relationship between edge data and feature data in machine learning. [14] explored how to use features to aid spectral clustering. [15] and [16] use edges and features that contain orthogonal information to better understand the relationship between the two.

Much of the work quantifying the expressive power of GNNs is achieved by relating GNNs to the classical Weisfeiler-Leman (WL) heuristic for graph isomorphism [17]. The WL test iteratively assigns node colors based on neighbors' colors until the partition generated by the node colorings remains stable [18]. Since WL aggregates information from neighboring nodes to determine a node's new color, WL can be thought of as a general message passing mechanism. In this way, the distinguishing capability of message passing GNNs can be directly related to the distinguishing capabilities of the WL algorithm. Higher order WL-algorithms also have been developed which consider collections of nodes as opposed to single nodes. These have inspired corresponding GNN architectures that have increased distinguishing capabilities [19].

2.3 Background

In this chapter, we evaluate the performance of the Graph Convolutional Network [20], Graph SAGE, the Graph Attention Network, and the Structure-Aware Transformer [21] on random graph models. We use orthogonal random clouds from the cSBM to append features to the degree-corrected SBMs and Poisson (the degree distribution for a normal SBM) SBMs to then use as our random graph models. The particulars of our random graph models and GNN architectures now follow.

2.3.1 Stochastic Block Models. The stochastic block model is a random graph model that encodes community clusters (equivalently, classes) in the graph topology. Edges are drawn independently from a Bernoulli distribution with probability determined by the class identities of the nodes. The symmetric SBM is a special case of the SBM wherein all classes have the same size and all inter- and intra-class edge probabilities are respectively uniform. The parameters for a symmetric SBM are: the total number of nodes n, the number of equally sized classes k, the intra-class edge probability p_{in}, and the inter-class edge probability p_{out}. While SBMs generate realistic clustering patterns, without further modification they exhibit a Poisson degree distribution. To more closely model certain data, [10] proposed the degree-corrected SBM which exhibits a heavy-tailed power law degree distribution.

In order to evaluate GNNs on SBMs, we must generate attribute information for each node in the graph. [12] proposed the contextual SBM (cSBM) to accomplish this task. In their paper, they investigate the properties of a specialized SBM by varying edge and feature similarity. They vary the edge similarity via a parameter λ, referred to as the edge information parameter. This is accomplished by assigning p_{in} and p_{out} as shown below where d is the desired average node degree:

$$p_{\text{in}} = \frac{d + \lambda\sqrt{d}}{n}, \quad p_{\text{out}} = \frac{d - \lambda\sqrt{d}}{n}$$

Setting λ equal to zero indicates there is no difference between inter- and intra-class edge probabilities, meaning the topology of the graph encodes no information about class groupings. A negative λ indicates that nodes of different classes are more likely to connect than nodes of the same class (heterophily), while a positive λ indicates the reverse relationship (homophily).

In the cSBM, features are assigned using Gaussian point clouds that are generated at a specified distance μ from the origin. Features, f, are thus defined as $f = \mu\mathbf{m} + z$, where f is a random variable, z a standard normally distributed random variable, and $\mathbf{m}$ the mean for the given class (all means are orthogonal in n-dimensional space). We can then vary the level of feature separability (feature information) by varying μ. A feature separation of $\mu = 0$ indicates node features are indistinguishable across classes, while a large value of μ indicates high distinguishability (measured by performance of neural networks). Thus we refer to μ as the feature information parameter.

A common task in graph learning is to recover the initial cluster assignment of each node. Current theory, such as [6], emphasizes strong recovery (an architecture's ability to correctly partition the entire graph). While such work helps us to understand the theory of GNNs, there has been much less work done on weak recovery (an architecture's ability to classify nodes better than chance) and even less on the transition between the two. We provide experimental data across all recovery regimes as a basis for future theoretical work. Understanding all ranges of recovery from weak to strong recovery is essential to identifying what variables directly contribute to an architecture's performance.

2.3.2 Graph Neural Networks. As stated before, we analyze the performance of four architectures: GCN, SAGE, GAT, and Transformer. Additionally we also analyze the performance of a standard feedforward neural network, which ignores the graph topology, and spectral clustering [22], which ignores node feature information [23].

We selected these specific architectures due to the diverse methods each uses to learn from data. For instance, the GCN performs an aggregation step in each layer by aggregating the features of each node's neighbors prior to passing the features through a shared feedfoward neural layer. The GAT performs a similar process with the addition of an attention mechanism to weigh each node's contribution prior to aggregation. SAGE aggregates along random walks to produce an embedding that contains more global information. Lastly, the Graph Transformer generates subgraph structure embeddings for each node prior to running a transformer [24] over the features and embeddings.

2.4 Experimental Design

To test the architectures, we fit a cSBM with average degree $k = 10$, the number of nodes $n = 1000$, the number of features $l = 10$, the number of classes $c = 2$ and the standard deviation of the Gaussian clouds to be 0.2. We chose these hyperparameters to be representative of a large variety of datasets without being too computationally expensive, e.g. 1000 nodes is large enough to get statistical regularity. Each of our architectures contains one input layer, a hidden layer of size 16, and an output layer (with ReLU activation functions). We use an Adam optimizer with a learning rate of 0.01 to train each architecture. As baselines, we trained a feedforward neural network (with one hidden layer of size 16) on the feature data and ran a spectral clustering algorithm on the edge data using sklearn [25].

With these hyperparameters, we vary λ between -3 and 3 and vary feature separation (cloud distance from origin) 0 from 0 to 2 to obtain 121×200 possible sets of graph data that range from highly disassortative to highly assortative (highly assortative is defined here as where spectral clustering or a neural network can distinguish between classes). Each of the GNN architectures were then trained on the synthetic data for 400 epochs (typically where the model ceased improving). We tested this model on a separate graph, with the same hyperparameters to prevent overfitting. We then plotted the average and maximum accuracy values over 10 trials with randomly generated seeds to ensure validity (see Figure 2.1). The number of graph datasets totals $121 \times 200 \times 10 \times 2 = 484,000$ for each architecture. Additionally, we highlight the areas where GNNs outperform solely feature or edge-based methods by outlining the regions where the difference in accuracy was greater than .5%.

We ran similar tests on graphs with 10 and 20 times the node count. Since the results of these tests showed no visual difference, we maintained a constant node count of 1000. Additionally, we passed a 5×5 Gaussian convolutional filter on the images to make them smoother. The code for this is posted in our Github repository.

In addition to the class count of two, we ran the architectures across class counts of three, five, and seven each with both a degree-corrected case and a Poisson case. Since each test was averaged over 10 trials, the number of tests totals 320 different tests with $15,488,000$ accuracy scores generated (more than .25 petaflops used in total).

2.5 Results and Discussion

In Section 2.5.1, we present results from our first (and simplest) set of experiments in detail to illustrate some preliminary interpretations. Then in Section 2.5.2 we compare performance across each of the four architectures. In Section 2.5.3 we compare the degree-corrected and non-degree-corrected cases. We then report what effect the increase in number of classes has on the architectures in Section 2.5.4. Lastly, we compare random models to real world data (in terms of GNN performance) in Section 2.5.5. For additional results and other noteworthy comparisons, see Appendix A. See also full code online to view additional architectures' performances: `https://github.com/brownthesr/Synthetic-Graphs.git`

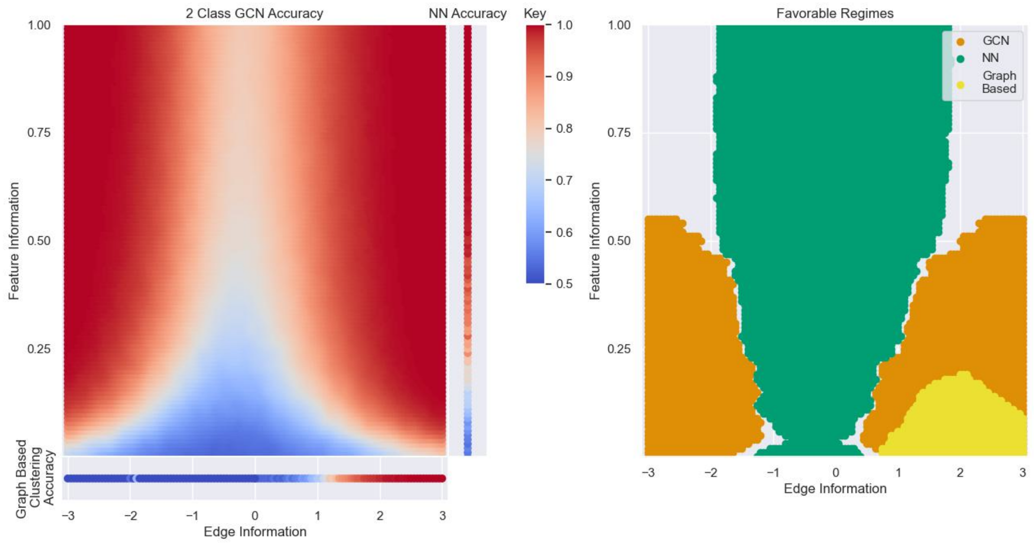

Figure 2.1: (Left) Performance of GCN on a two-class non-degree-corrected cSBM. Color indicates accuracy. The performance curves for the feedforward neural network and graph-only (feature-blind) architectures are also represented to the right and below respectively, each one dimensional because they only use one aspect of the graph (edges or features). We used spectral clustering for assortative graphs (where edge information is between 0 and 3) whereas we used graphtool in the dissasortative case (edge information is between -3 and 0) where spectral clustering doesn't make sense. We ran additional experiments for other clustering algorithms using the Python package Leiden [1]. However, since they each performed worse than spectral clustering, we do not include them here. (Right) The top-performing model (between GCN, NN, and spectral clustering). White space indicates where one model was not consistently better than the others. We see that GCN is favorable when both edge and feature information is moderately noisy. The other two models surpass the GCN when one aspect of the graph is relatively noisy and the model is able to completely ignore it. Additionally, we see that the GCN also outperforms the other models if the graph is sufficiently heterophilous.

2.5.1 GCN Performance on Binary Node Classification. Figure 2.1 displays the average accuracy of our experiment with a two-class GCN on a spectrum of cSBMs.

Studying the areas where the edge or feature information is zero is informative of how well the model utilizes only one of these information sources (edge or features) in isolation. In cases where zero edge information exists, enough feature information can improve GCN performance. This is possible because of the implicit self edges in a 2-layer GCN, wherein a diffused form of the nodes' original attributes returns to the GCN. However, given zero feature information, it is impossible for the model to achieve better-than-random recovery. These results illustrate that GCN cannot classify based on edge information alone, but rather edge information enhances the feature information.

The phase transition (the color change from blue to red) is not linear. The fact that the red portion dips inward demonstrates that when edge and feature information are used at the same time the model performs better than with either separately. Furthermore, by viewing where the GCN performs the best (the dark orange area on right), we can see what minimum corresponding levels of features/edge information are needed for the GCN to outperform their corresponding feature/edge-agnostic counterparts, a "favorable regime." Conversely, the exterior of these regions,where the plot is not dark orange, shows that in many cases, simply utilizing one set of information (feature or edge) is preferred over using both simultaneously with the GCN.

Lastly, it is possible for the model to utilize edge information even if it is heterophilous (indicated by negative edge information values). In the next section we illustrate that as we increase the number of classes, the GCN struggles with heterophilous networks. Interestingly, the favorable regime on the heterophilous side is larger than the homophilous side (the dark orange section on the left vs the right) due to lack of a graph-only model that can cluster bipartite graphs effectively. This could also imply that GCNs have more potential for processing heterophilous graphs than previously thought [26].

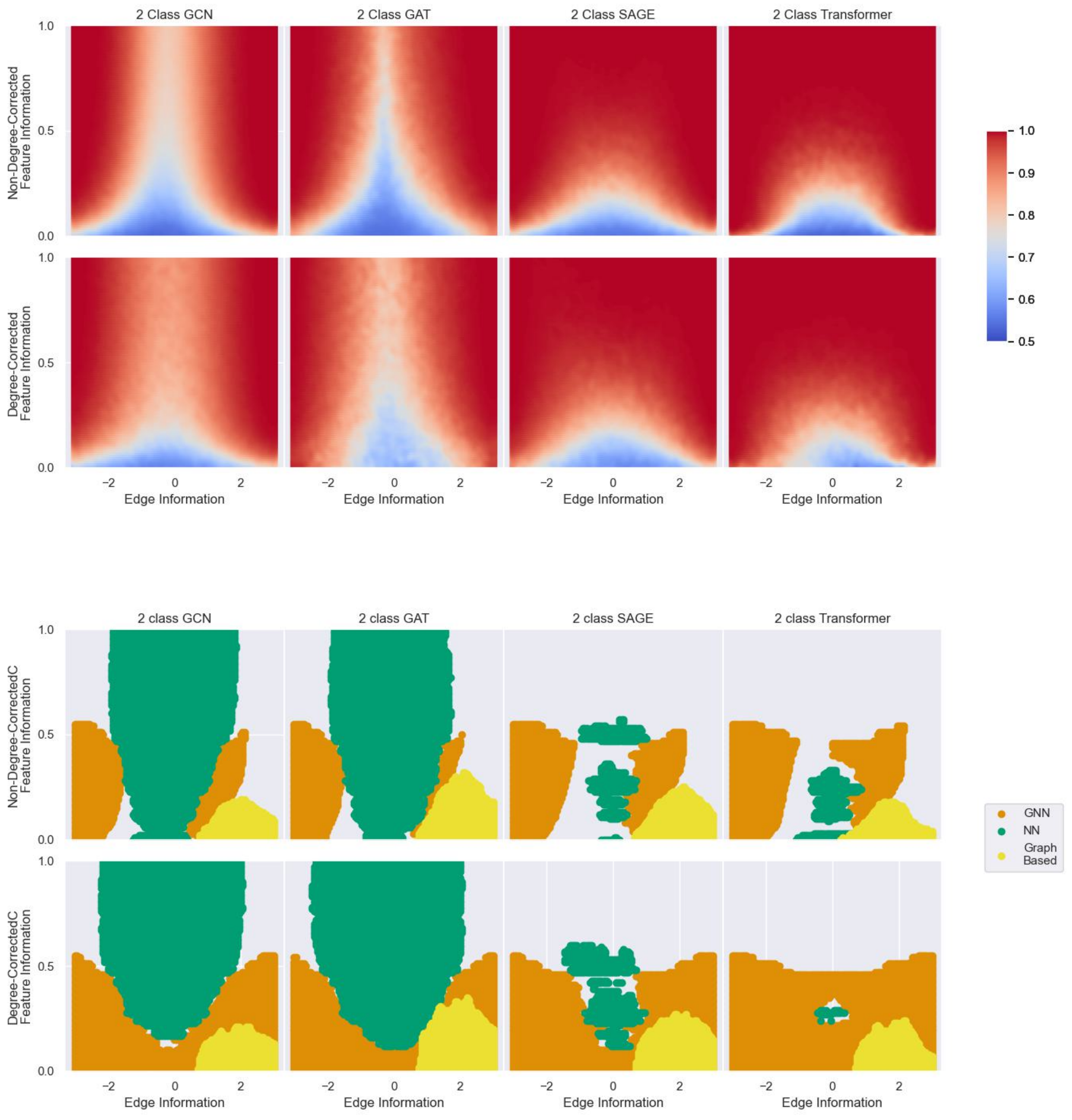

Figure 2.2: Comparison of architecture performances on non-degree-corrected and degree-corrected SBMs for GCN, GAT, SAGE and Transformer Architectures. Notice the GCN and GAT consistently perform worst when the edge information is roughly zero, but the other two models are able to still achieve perfect accuracy given enough feature information. This perhaps could be due to SAGE and Transformer being able to learn a more global context for each node. In this regime we see that almost all of the architectures did better on the heavy tailed graphs. The GCN was able to achieve higher higher accuracy when the edges were just noise. The accuracy of the GAT improved in the regime of very noisy edges and features, this can bee seen by viewing the lightness of the blue portion.

2.5.2 Performance of GCN, GAT, SAGE, and Transformer Architectures. In Figure 2.2 we compare the performance of the GCN, GAT, SAGE, and Transformer on a two-class, non-degree-corrected cSBM. We see in the bottom of Figure 2.2 that SAGE is rarely beat by a neural network, showing that it is resistant to feature noise compared to the GAT or GCN. In addition, SAGE and Transformer achieve perfect classification with zero edge information, something that the GCN and GAT fail to achieve. This reflects that SAGE utilizes more global information (random walks and graph embeddings) than the GCN and GAT, and that the Transformer learns to ignore the graph embedding.

The areas where the models perform the worst (blue areas in top half of Figure 2.2) vary across the architectures, as do the regions where they outperform the neural network and spectral clustering baselines (bottom half of Figure 2.2). For example, the GATs blue portion goes much higher than the other architecture in the regime with zero edge information. This suggests that clean features are especially important to the GAT. In addition, curiously, the degree correction seems to help both the Transformer and the GAT perform slightly better in the heterophilous case. Notice how the blue on those graphs is slightly skewed right. In addition, all of the models perform better on degree corrected graphs.

2.5.3 Degree-Corrected SBMs. When using degree-corrected SBMs, we observed similar changes in the performance of the various architectures.

We note that the GCN appears to perform better on heavy-tailed data. We believe this occurs because most nodes in the heavy-tailed data will have relatively few neighbors, allowing for fewer confusing neighbors to contribute misleading information in the aggregation step than in the Poisson degree distribution. This is similar to ideas from [27].

Improved performance was consistent across all models, for example the performance of SAGE only increased towards the edges of its blue region. The performance of GAT increased dramatically in the case of very noisy edges and features. This is likely because GAT was already pruning bad edges, So perhaps degree correction gave it more information on what edges to prune. Interestingly, the attention based models, the Transformer and GAT, saw a starker increase in performance in the heterophilous case with degree corrected graphs than in the homophilous case.

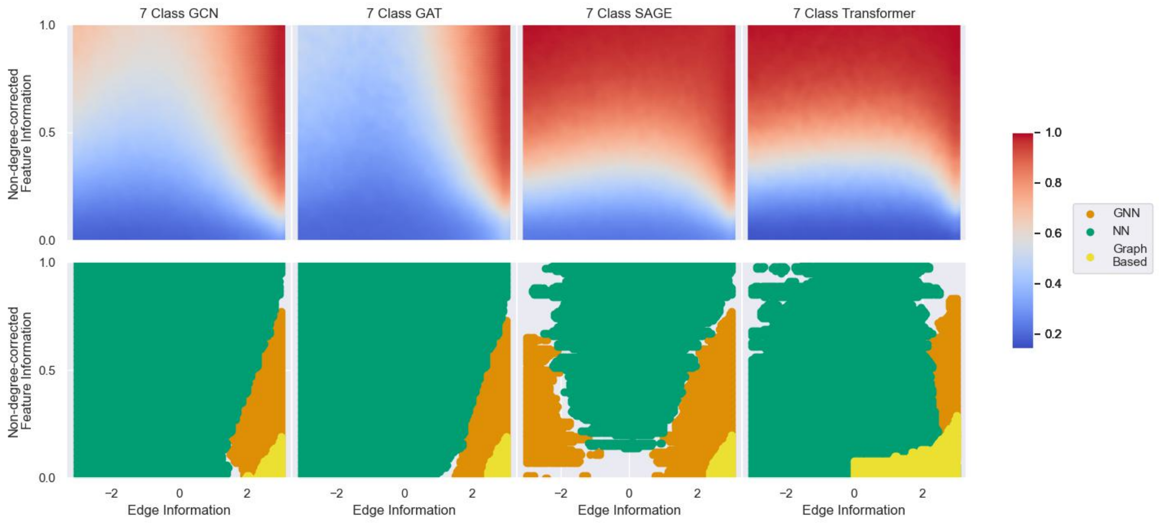

Figure 2.3: Architecture performance on data with seven classes. Here the models need more feature or edge information than in binary classification. This is similar to results in [2], where an increased number of classes necessitates more distinct clusters in order to accomplish recovery (see page 16). Viewing how these blue regions shift illuminates how well each model handle heterophily. SAGE consistently did the best both when increasing the number of classes and in the heterophilous cases.

2.5.4 Performance with seven classes. In addition to degree distribution, we also assessed what effect increasing the number of classes had on the various architectures. These results are summarized in Figure 2.3. A more careful comparison is presented in Appendix A.

As we increased the number of classes, we found that in the heterophilous regime (the left side of the graphs in Figure 2.3) GCN and GAT progressively performed worse. The GCN is able to achieve partial recovery if we give it enough feature information (this is more apparent in Appendix A), whereas the GAT's performance did not improve as much with more feature information. This contrasts to the performance of SAGE and Transformer which were able to retain performance far better in the heterophilous case.

So heterophily can be especially harmful for GCN and GAT [4]. This stark contrast in performance could be due to GCN and GAT not learning to completely ignore harmful edges. This is consistent with ideas from [6]. It could also be that SAGE and Transformer are just able to learn a higher level of complexity across the spectrum of graphs since they learn a global embedding.

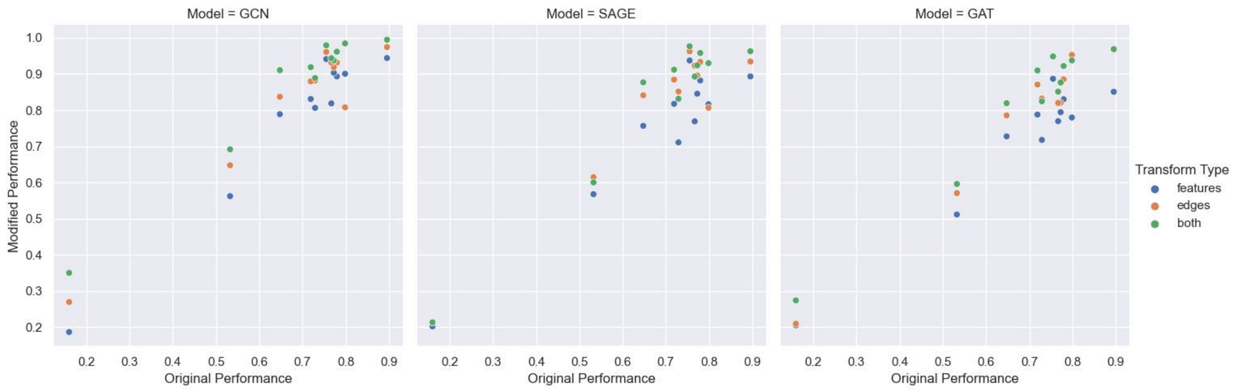

Figure 2.4: Comparison between model accuracies on real data compared to performance on matched synthetic data for GCN, SAGE and GAT on standard benchmark datasets. The datasets from left to right are: Flickr, DeezerEurope, Citeseer, LastFMAsia, DBLP, FacebookPagePage, Pubmed, GitHub, Cora, Amazon Computers, and Amazon Photos. The figure depicts cases where we transform only the edges, only the features, and both. The transformer was not run due to memory requirements. The accuracy tends to improve when we transform the data, the most stark increase being when all the data is transformed. This is likely due to the new edges and features more faithfully representing the classes than did the original data.

2.5.5 Connection to Real World Datasets. In order to assess the difference in performance of the architectures on synthetic data versus real world data, we generated matched cSBM versions of several benchmark datasets and compared how the models' accuracy changed between the original and cSBM. We found that on cSBM versions of the benchmark datasets, the models generally performed better than on the original versions.

To make the cSBM as authentic as possible, we transformed the edge and feature data as if each dataset was already a degree-corrected cSBM. The feature data was transformed by calculating the mean and standard deviation of the present features according to classes, then sampling independently from a Gaussian distribution. The edge data was randomized such that the empirical degree distribution and inter/intra-class edge probabilities were maintained, following [28].

Using this method of transforming the edge and feature data to make a cSBM, we record the performance of a GCN on both the unscrambled and scrambled version of 11 real world datasets. We use the following datasets obtained through pytorch geometric: Flickr [29], DeezerEurope[30], Citeseer [31], LastFMAsia [30], DBLP [31], FacebookPagePage [32], Pubmed [31], GitHub [32], Cora [31], Amazon Computers [33],and Amazon Photos [33].

Scrambling a dataset had a positive impact on the accuracy of the GCN if the labels are known beforehand (see Figure 2.4). Indeed, performance is boosted slightly by scrambling solely the features, but significantly boosted by scrambling solely edges.

It is likely that the gap between the transformed and original data has something to do with more complex substructures in the empirical graph data, which would be erased by the scrambling process. When we scramble the edge data, we remove all statistically consistent structure within the graph except degree and community structure. Hence, substructures like triadic closure or motifs are completely erased. The fact that the GNNs do better on randomized data suggests that they may perform optimally on SBM-like data, but are negatively impacted by the additional structure present in real data. Uncovering why such structure is detrimental to these GNNs is a significant opportunity for future work.

2.6 Future Directions

Future directions include investigating analytical bounds for weak recovery across the various architectures. Analytically treating the observed phase transitions would be beneficial to the geometric learning community as it could lead to better architectures and a more concrete understanding of various GNN recovery thresholds. Additionally, proving theoretically what type of graph topology is optimal for GNN performance could lead to development of better architectures and improve our a priori ability to match existing architectures to specific problems. We also saw that, among the architectures we tested, GNNs performed better when complex structures were erased. Developing a model to more effectively incorporate higher order edge structure would be a great avenue of future research. Lastly, while we saw that some of the architectures (SAGE and Transformer) were able to achieve perfect accuracy with uninformative edges, there was no architecture able to achieve perfect accuracy with uninformative features by utilizing solely the edges. Developing an architecture to fully interpolate between solely edge- and feature-based methods would lead to better GNN performance across a wider variety of graphs.

Acknowledgements

This chapter is a joint work between me, Drake Brown, Zach Boyd, and Cole Hanson. We would like to thank the Office of Research Computing at BYU for providing the computing power necessary for this project. We would also like to thank ZiRui Su and Ryan Wood for their contributions.

Chapter 3. A Theoretical Approach to Weak Recovery in GNNs

In this chapter, we seek to articulate in theoretical terms the connection between the parameters of a cSBM and the performance of GCN architectures. Some work has already been done applying SBMs to the theoretical understanding of GNNs. [6] demonstrates thresholds at which an edge attention mechanism such as the one used in GAT [7] is able to separate two classes based on edge densities and feature separation. [34] demonstrates performance guarantees in the training process for a GCN. Both works represent substantial contributions to understanding GNNs performance theoretically, however neither yields an explicit formula for predicting directly the probability of a correct classification prediction.

In our analysis, we will seek to demonstrate such an explicit formula and directly compute the probability that a GCN correctly classifies a chosen node given a predetermined set of parameters. We will also walk through the mechanics of deeper (2+ layer) GNNs and analyze the role of the nonlinear activation function in separating the data. We follow suit with the previous approaches in assuming all classification problems mentioned hereafter are binary.

3.1 Single-Layer GCNs

We begin with perhaps the simplest GNN architecture out there: a single layer GCN. The equation for this model is

$$f_W(X) = \sigma\left(AXW\right)$$

Where W is a $N_{\text{feat}} \times 2$ matrix of weight parameters, A is the $N \times N$ adjacency matrix of the graph G (with added self-loops), X is the $N \times N_{\text{feat}}$ matrix of node features, and σ is an order-preserving nonlinearity, conventionally assumed to be the softmax activation function defined componentwise by the expression:

$$\sigma(x)_i = \frac{\exp(x_i)}{\sum_{j=1}^{k} \exp(x_j)}.$$

We may also rewrite the node-level equation for this model as follows:

$$f_W(X)_i = \sigma\left(\sum_{j\in N(i)} X_j W\right).$$

3.1.1 Fixed Graphs with Gaussian Features. To prove a result connecting the parameters of an SBM to the prediction accuracy of the single layer GCN, we first consider a fixed graph $G = (V, E)$ with adjacency matrix A (which we assume has added self-loops for each node), and we choose a fixed node $i \in V$. We assume that the weight matrix W is already trained. We assume that $2 \mid N$, so that exactly $N/2$ of the nodes are assigned to class 1 and $N/2$ of the nodes are assigned to class 2. Without loss of generality, we assume that node i belongs to class 1.

For node features, we choose an N_{feat}-dimensional unit vector m to serve as the mean for the Gaussian point cloud corresponding to class 1, and we let $-m$ be the mean for the point cloud corresponding to class 2. We let $\mu > 0$ be the feature separation parameter. Supposing that the means are diametrically opposed makes for simplicity in downstream calculations; it is worth noting that all choices of 2 means may be translated to fit this assumption. We thus choose each node j's feature vector as follows:

$$X_j = \begin{cases} \mu m + z_j & \text{if node } j \text{ is in class 1} \\ -\mu m + z_j & \text{if node } j \text{ is in class 2} \end{cases}$$

where z_j is drawn i.i.d. from an n-dimensional standard normal Gaussian distribution. With the adjacency, features, and weights determined we are now ready to begin our analysis.

We begin by addressing the nonlinearity σ. We consider the outputs of f_W. Since our classification task is binary, the node-level outputs $f_W(X)_j$ represent node embeddings in the plane $\mathbb{R}^2$. Assuming the conventional approach to classification, our model predicts

$$\hat{Y}_i = \arg\max\left(f_W(X)_i\right).$$

In other words, the index of the largest component of $f_W(X)_i$ is the final class prediction of node i. Since σ is assumed to be an order-preserving function, the final class prediction of a node is determined simply by the nested expression

$$\sum_{j \in N(i)} X_j W.$$

Here $N(i)$ is the 1-hop neighborhood of node i including node i itself. From here, we split the sum into two parts corresponding to the two possible classes of neighbors:

$$= \sum_{\substack{j \in N(i) \\ j \text{ in class 1}}} X_j W + \sum_{\substack{j \in N(i) \\ j \text{ in class 2}}} X_j W$$

We then substitute the known expressions for X_j:

$$= \sum_{\substack{j \in N(i) \\ j \text{ in class 1}}} (\mu m + z_j) W + \sum_{\substack{j \in N(i) \\ j \text{ in class 2}}} (-\mu m + z_j) W.$$

Then letting n_1 and n_2 be the number of neighbors of i in class 1 and class 2 respectively, we obtain:

$$= \underbrace{\mu(n_1 - n_2) m W}_{\text{constant}} + \underbrace{\left(\sum_{j \in N(i)} z_j \right) W}_{\text{randomness}}.$$

Notice that all of the randomness is siloed into the second term of this expression. Also note that $n_1 \geq 1$ since we assumed that node i is a neighbor to itself. We set $z' = \left(\sum_{j \in N(i)} z_j \right) W$ for notational convenience. Note now that since each z_j is standard normally distributed, we have that z' is also normally distributed with mean 0, and covariance matrix $|N(i)| W^T W$.

From this point we may calculate the probability that the model correctly classifies node i as class 1 by integrating the probability density function of $\mu(n_1 - n_2) m W + z'$ over the region $\mathbb{R}^2_{x>y}$. We may equivalently compute this integral by finding the orthogonal distance δ of the Gaussian mean $\mu(n_1 - n_2) m W$ from the decision boundary $y = x$, shifting the mean to the origin, and rotating clockwise 45 degrees to obtain the integration problem illustrated in Section 3.1.1.

Formally, the covariance matrix of the transformed distribution is given by

$$\Sigma = |N(i)| R^T W^T W R$$

where R is a 2×2 matrix describing a 45° clockwise rotation of $\mathbb{R}^2$. The distance δ is

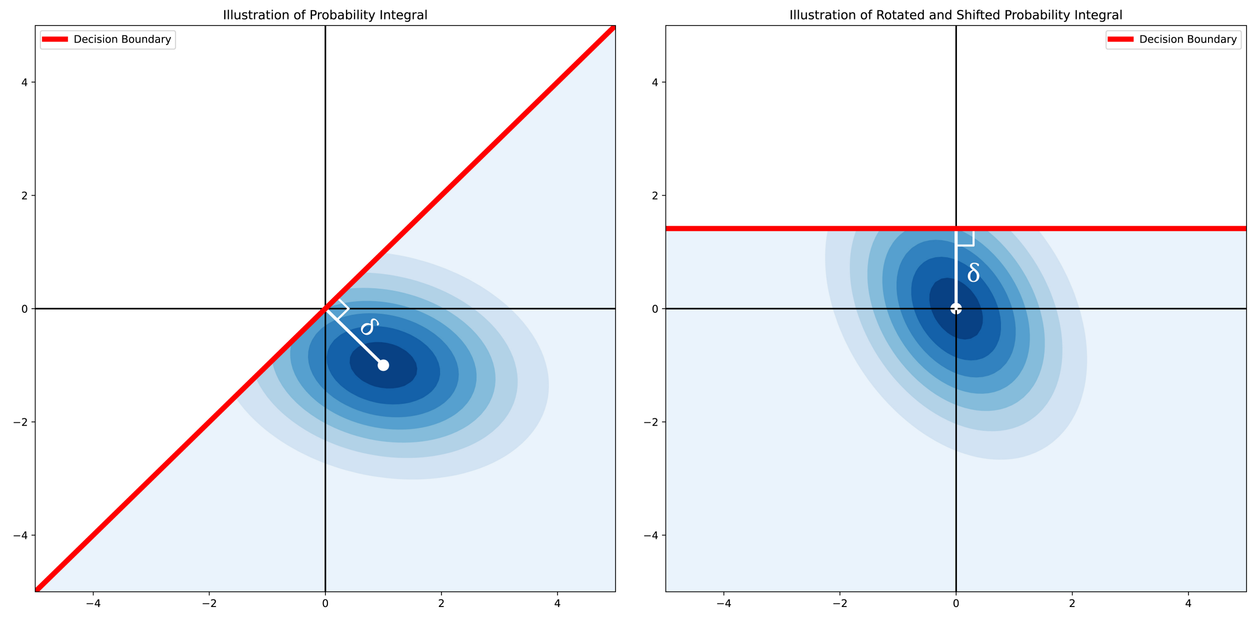

Figure 3.1: Plot of the integration problem before and after transformation.

described by:

$$\delta = \operatorname{sgn}(n_1 - n_2)\left\|\mu(n_1 - n_2)mW - \operatorname{proj}_{(1,1)}\left(\mu(n_1 - n_2)mW\right)\right\|.$$

Note that the sign of δ matters for computing the probability of a node being placed in class 1. Thus the Gaussian probability density function we seek to integrate is given by

$$\operatorname{pdf}(x, y) = (2\pi)^{-1}\left(\det \Sigma^{-1/2}\right)\exp\left(-\frac{1}{2}(x, y)\Sigma^{-1}(x, y)^T\right)$$

assuming implicitly that W^TW is invertible. This assumption is reasonable since real invertible 2×2 matrices are dense in the space of real 2×2 matrices, and in practice W is trained by gradient descent and is not likely to have W^TW singular. Therefore the integral we seek to compute is given by

$$\int_{-\infty}^{\delta}\int_{-\infty}^{\infty} \operatorname{pdf}(x, y)dxdy.$$

This leads us to our first result. A full derivation is given in the appendix.

Theorem 3.1. *The probability that f_W predicts node i is in class 1 is given by:*

$$\frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{\delta}{\sqrt{2\sigma_y}}\right)\right)$$

Where

$$\Sigma = \begin{pmatrix} \sigma_x & \sigma_{xy} \\ \sigma_{xy} & \sigma_y \end{pmatrix}$$

Is the covariance matrix of the rotated and shifted distribution described above, and erf *is the Gaussian error function.*

3.1.2 Extension of Result to SBMs. We now apply the result on fixed graphs to connect the parameters of SBMs to single layer GCN performance. Assume now that G is drawn from a 2-class SBM with n nodes, average degree d, and

$$p_{\text{in}} = \frac{d + \lambda\sqrt{d}}{n}, \quad p_{\text{out}} = \frac{d - \lambda\sqrt{d}}{n},$$

where λ is the edge information parameter as described in the previous chapter. Recall the node-level equation for the output of a single-layer GCN:

$$f_W(X)_i = \sigma\left(\sum_{j\in N(i)} X_j W\right).$$

Also recall that the classification result is completely determined by the nested expression

$$\sum_{j\in N(i)} X_j W = \mu(n_1 - n_2)mW + \left(\sum_{j\in N(i)} z_j\right) W$$

Hence we have, taking the expectation conditioned on the z_j with respect to the randomness of the edges of G,

$$\mathbb{E}\left[\mu(n_1 - n_2)mW + \left(\sum_{j\in N(i)} z_j\right) W\right] = \mu(\mathbb{E}[n_1] - \mathbb{E}[n_2])mW + \left(\sum_{j\in N(i)} z_j\right) W.$$

Now we compute $\mathbb{E}[n_1]$ and $\mathbb{E}[n_2]$, or the expected numbers of neighbors of i belonging to class 1 and class 2 respectively. Remember that the node of interest, node i, belongs to class 1. Therefore,

$$\mathbb{E}[n_1] = p_{\text{in}} \cdot \left(\frac{n}{2} - 1\right), \quad \mathbb{E}[n_2] = p_{\text{out}} \cdot \frac{n}{2}.$$

Then we have

$$\mu(\mathbb{E}[n_1] - \mathbb{E}[n_2])mW + \left(\sum_{j\in N(i)} z_j\right) W$$

$$= \mu \left(p_{\text{in}} \left(\frac{n}{2} - 1 \right) - p_{\text{out}} \cdot \frac{n}{2} \right) mW + \left(\sum_{j \in N(i)} z_j \right) W.$$

Note that taking the expectation with respect to the randomness of the edges only affects the location of the distribution mean. In the integration problem illustrated before, this means that only δ is affected by edge randomness. Hence to generalize the result we need only update the value of δ to its expected value with respect to edge randomness in the SBM. Doing so yields the probability that node i is classified correctly assuming that G has the expected edge distribution. More work may be necessary to determine the exact overall expected probability of correct classification.

3.1.3 Experimental Validation. Figure 3.2 illustrates experimental results for node-level probabilities on actual SBM-drawn graphs. The figure demonstrates that the probability predictions obtained via Theorem 3.1 capture the effects of varying graph topology. We note that these results are examples from 2 graphs which we determined to be representative of the phenomena observed.

3.2 Two-Layer GCNs

We will now add complexity to the model by introducing a second convolutional layer to our GCN. We now write the equation for a simple two-layer GCN architecture as follows:

$$f_{W_1,W_2}(X) = \sigma_2 \left(A\sigma_1 \left(AXW_1 \right) W_2 \right)$$

where σ_1, σ_2 are order-preserving nonlinearities, A is the $N \times N$ adjacency matrix for G with added self-loops, and W_1 and W_2 are $N_{\text{feat}} \times N_{\text{hid}}$ and $N_{\text{hid}} \times 2$ parameter weight matrices respectively. It is important to note that the 2-layer GCN introduces variability in the number of hidden feature dimensions N_{hid}.

3.2.1 Understanding How GCNs Manipulate Data. The two-layer case is substantially more complex than the one-layer case. We work step-by-step to understand how a

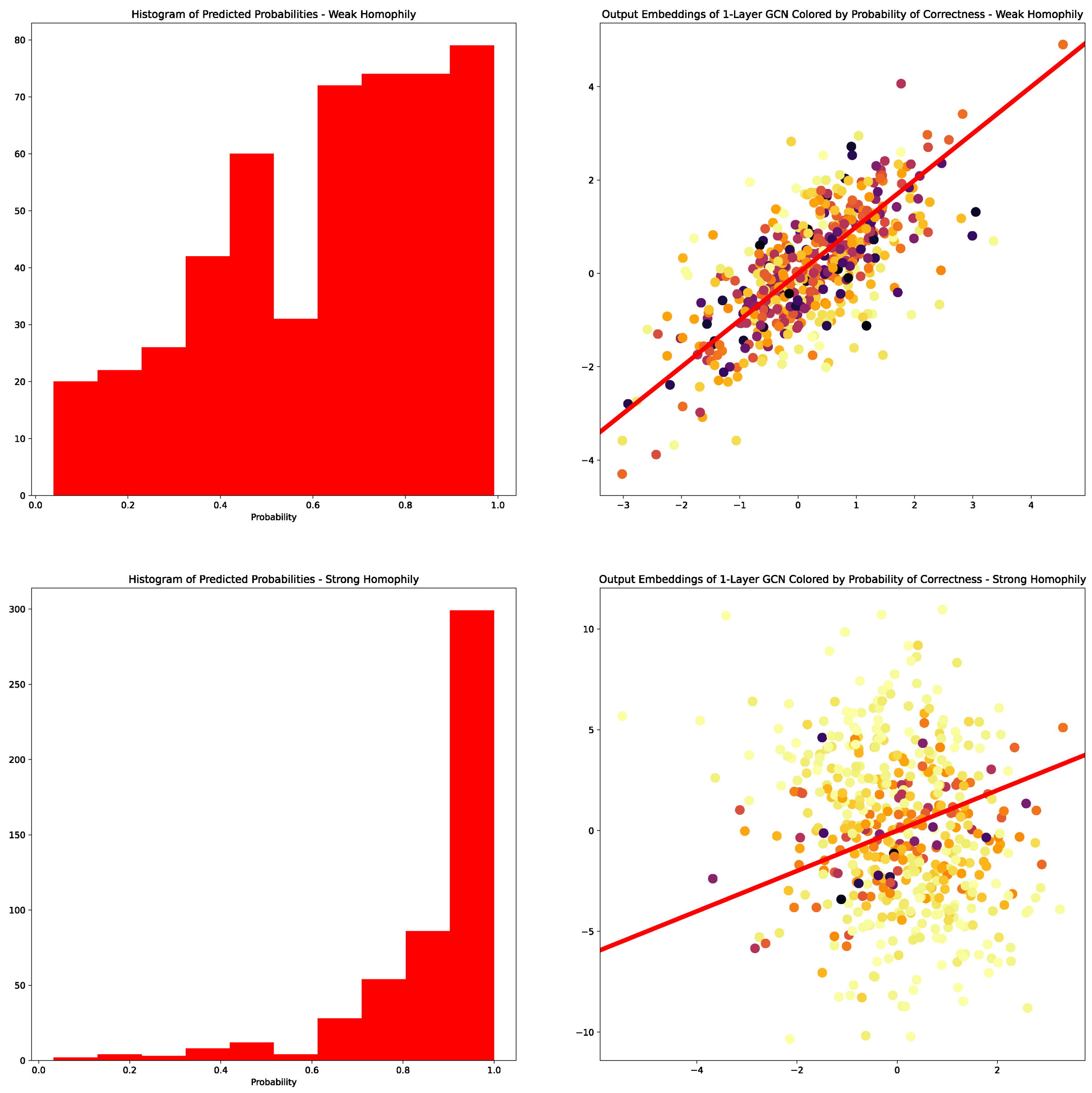

Figure 3.2: Theoretical probabilities computed on experimental SBM graphs ($n = 500$, $\mu = 1$, $d = 10$). Results from both weakly homophilous ($\lambda = 0.5$) and strongly homophilous ($\lambda = 1.5$) example graphs are demonstrated. Weakly and strongly heterophilous graphs exhibit similar performance respectively. Notice that predictions are far more certain in the strongly homophilous graph as opposed to the weakly homophilous graph. Also, as expected, uncertain output embeddings tend to be distributed closer to the decision line $y = x$ in both cases.

two-layer GCN transforms the distribution of the inputs.

To do this we use an example GCN with $N_{\text{feat}} = N_{\text{hid}} = 2$ for the sake of visualization. We train this GCN on an SBM with $N = 500$ nodes, average degree $d = 10$, $\mu = 1$ for feature separation and $\lambda = 1$ for homophilous edge separation. We assume the node feature data are drawn from Gaussian point clouds corresponding to class means as before. The distribution begins as two diametrically opposed point clouds as illustrated in the figure:

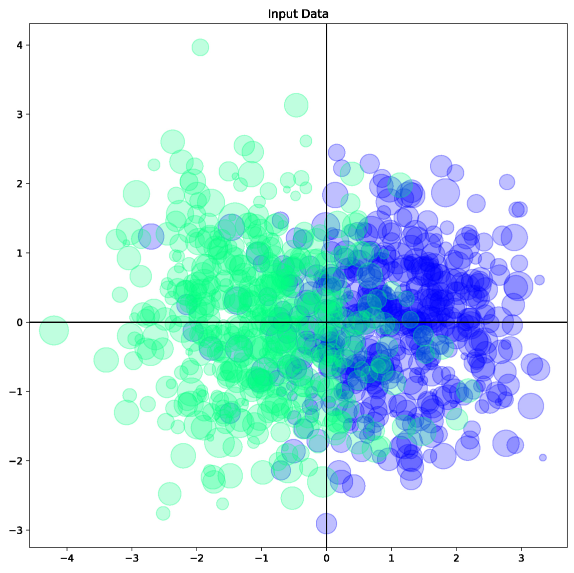

Figure 3.3: Example of input data distribution. Points are colored by class identity and sized by $|n_1 - n_2|$, the imbalance of the node's neighborhood toward one class.

As in the single-layer case, if node i is in class 1, then node i's feature data can be described mathematically by:

$$X_i = m + z_i$$

where z_i is drawn i.i.d. from a standard N_{feat}-dimensional normal distribution.

We then aggregate node data from neighbors. Under the homophily assumption $\lambda > 0$, each node is likely to have more in-class neighbors than out-class neighbors. The feature data of in-class neighbors is biased toward the in-class mean. Thus aggregation on homophilous graphs pulls node features in the direction of their associated class means. This results in greater separation and denoising of the Gaussian point clouds as illustrated by Figure 3.4.

Mathematically, first-layer aggregation and linear transformation yields an equation for the modified feature data of node i nearly identical to the single-layer GCN outputs:

$$\mu(n_1 - n_2)mW_1 + \left(\sum_{j \in N(i)} z_j \right) W_1$$

which, as before, follows a normal distribution with mean $\mu(n_1 - n_2)mW$ and covariance matrix $|N(i)|W^T W$.

It's important to note a key distinction between the heterophilous ($\lambda < 0$) and homophilous ($\lambda > 0$) cases. In the homophilous case, node i's features and its neighbor's features are biased towards the same class mean, whereas in the heterophilous case node i's features are biased in a direction opposite that of most of its neighbors (which lie in the opposing class). In other words, adding self-loops to the graph biases the model towards homophily. This difference would imply that the GCN needs slightly stronger heterophily than homophily to obtain similar levels of performance. In other words, swapping a positive edge parameter λ for $-\lambda$ will likely reduce classification accuracy. This observation may account for the asymmetric performance phenomena seen previously in Figure 2.1.

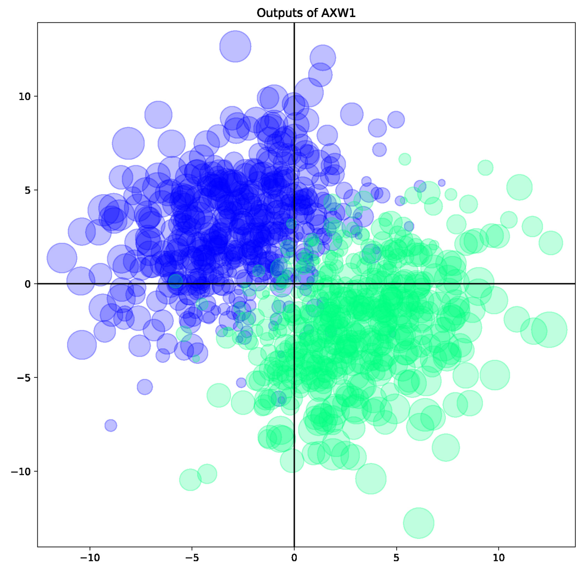

Figure 3.4: Example of data after the transformation AXW_1

The activation function σ_1 (which is ReLU in this example) is then applied to the data, which collapses the negative components of the data to 0. Note that in this example, the trained matrix places the clusters mostly in quadrants II and IV, which means that ReLU projects most points onto the corresponding axes, effectively isolating the class 1 and class 2 components of the embeddings. Similar phenomena occur in higher dimensions, suggesting that a trained GCN leverages the nonlinear activation function σ_1 to isolate meaningful subsets of features.

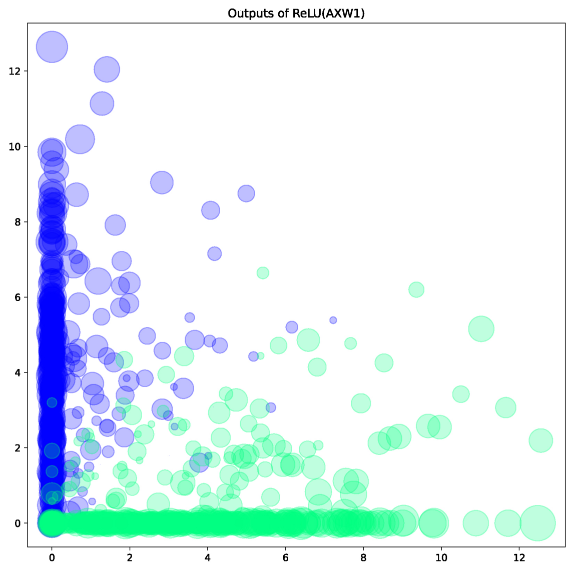

Figure 3.5: Example of data after the transformation $\sigma_1(AXW_1)$

The nonlinearity greatly increases the theoretical complexity of the problem. Using ReLU as an example, we now find that the transformed data

$$\sigma_1\left(\mu(n_1 - n_2)mW + \left(\sum_{j \in N(i)} z_j\right) W\right)$$

for node i follows a mixture of distributions which we will now describe. Let $\text{pdf}_i(x)$ be the Gaussian distribution function of node i's transformed data prior to applying the nonlinearity $\sigma_1 = \text{ReLU}$. Because ReLU collapses all negative components of the data to 0, each hyperplane on the boundary of the positive hyperquadrant $\{x \in \mathbb{R}^{N_{\text{hid}}} : x_j \geq 0 \, \forall j\}$ must be treated differently. To begin, each boundary surface corresponds to a subset of dimensions $S = \{k_1, ..., k_s\}$ which are fixed at 0. We choose a particular subset S with probability determined by

$$P_S = \int_{Q_S} \mathrm{pdf}_i(x)$$

The integral of $\mathrm{pdf}_i(x)$ taken over the quadrant Q_S corresponding to vectors in $\mathbb{R}^{N_{\text{hid}}}$ whose components $k_1, ..., k_s$ are negative and all others are positive. Then from the boundary surface corresponding to S, points are chosen with probability:

$$\int_{-\infty}^{0} \cdots \int_{-\infty}^{0} \mathrm{pdf}_i(x)\, dx_{k_1} \cdots dx_{k_s}.$$

We note that in the trivial case that $S = \emptyset$ then the above distribution function is simply $\mathrm{pdf}_i(x)$.

A second aggregation of node data is then performed by taking $A\sigma_1(AXW_1)$. With meaningful components of the data strengthened by the activation function, this second aggregation incorporates the denoised class information of neighboring points, resulting in a strong separation of the two transformed point clouds.

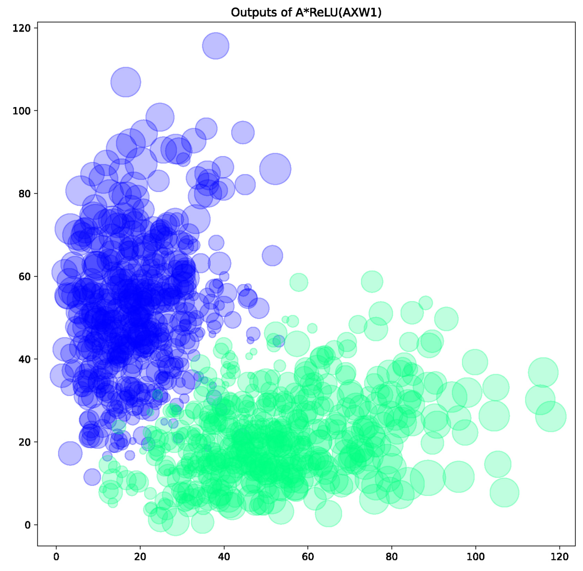

Figure 3.6: Example of data after the transformation $A\sigma_1(AXW_1)$

The second aggregation layer introduces significant theoretical complexity. Because each node's data has already undergone aggregation once, the layer 1 hidden embeddings are not independent of one another; nodes which share an edge will have hidden embeddings influenced by each other. Moreover, the nonlinearity fundamentally changes the distribution of the points such that a probability distribution function cannot be easily described. The distribution of the output of layer 2's aggregation function can be described loosely as the sum of neighbor node embedding variables drawn from the mixtures of distributions described previously. However, the mixture of distributions describing a particular hidden embedding of node $j \in N(i)$ depends on the balance of in- and out- class neighbors for node j. Imposing a strong homophily or heterophily assumption on the graph may result in greater regularity of these neighbors, however for the exploratory purposes of this chapter we will not delve here into greater detail.

Applying a final linear transformation to the data positions the two clouds optimally relative to the decision boundary. The difference between the separation of the final transformed clouds in Figure 3.7 and the original clouds in Figure 3.3 is stark. Note that prior to the final nonlinear order-preserving activation σ_2 (usually softmax), the final class prediction of each node is fully determined.

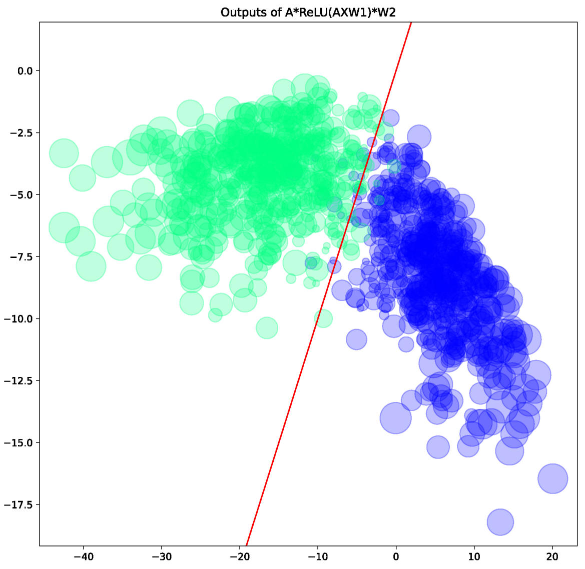

Figure 3.7: Example of data after the transformation $A\sigma_1(AXW_1)W_2$

3.2.2 Comparison to 2-Step Aggregation Linear Model. In the previous subsection we illustrated how a GCN is able to separate point clouds using neighborhood aggregation and nonlinearities. We now pose the question: is aggregation without the nonlinearity enough to achieve comparable classification accuracy? To better understand the role of the nonlinearity σ_1 in a 2-layer GCN, we experimentally tested the 2-layer GCN f_{W_1,W_2} against a linear model $g_W(X) = A^2XW$. Notice that removing σ_1 from the GCN yields a model which flattens into g_W. Training both models with the same loop on the same data, we found that in almost every case the GCN attained a significantly higher classification accuracy percentage than the 2-step linear aggregator model.

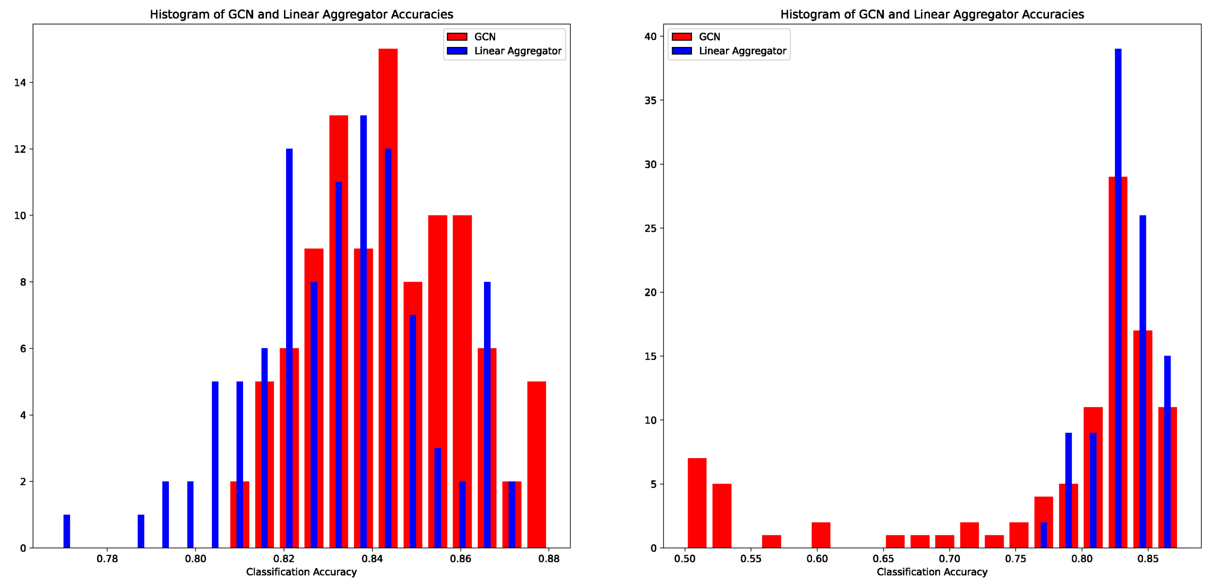

Figure 3.8: Comparison of accuracy test results for the 2-layer GCN and the 2-step linear aggregator. The histogram on the left compares the results of a GCN with $N_{\text{hid}} = 10$ and a 2-step linear aggregator. The histogram on the right compares the results of a GCN with $N_{\text{hid}} = 2$ and a 2-step linear aggregator.

The output embeddings from the GCN and the linear aggregator are substantially different. Note that the nonlinearity of the GCN allows for the model to smash inter-cluster noise more effectively onto the correct side of the decision line. The 2-step linear aggregator model cannot fundamentally reshape the distribution of the data; the output data is still necessarily Gaussian. While the 2-step aggregation filter does substantially reduce noise in the data,

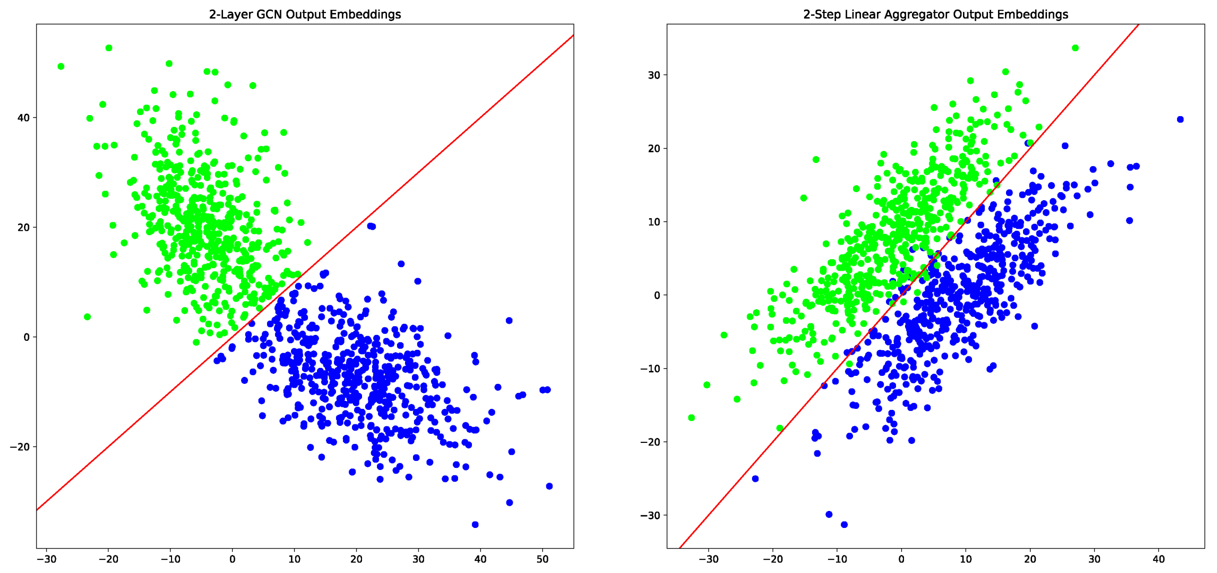

Figure 3.9: Comparison of output embeddings for the 2-layer GCN and the 2-step linear aggregator.

the linear model cannot isolate meaningful components of the data like the GCN can. This agrees with the view in [9] of GNNs as more effective noise filters, and offers more insight into how denoising is achieved.

3.2.3 Key Observations on 2-Layer GCN Mechanics. The 2-layer GCN obtains its representational power through the nonlinearity after the first layer. In particular, ReLU offers the substantial ability for the model to isolate specific components of the data. The second layer aggregation then strengthens those component signals resulting in a thoroughly denoised output.

3.3 Future Directions

In this chapter, we obtained a theoretical prediction for the probability of a chosen node from a 2-class SBM being correctly classified by a single-layer GCN. We also walked through the mechanics of a 2-layer GCN and demonstrated how denoising of the input data is impacted by the usage of a nonlinearity between aggregation layers. This work represents a novel approach to quantifying weak recovery and predict node-level accuracy of a GNN architecture theoretically, and a first demonstration of how GCN separates normally distributed data.

There are many exciting opportunities for further theoretical research in GNN performance analysis. Here are just a few:

- stronger bounds for the 2-layer case,
- alternative formulations for weak recovery results,
- weak recovery analysis for other popular GNN architectures or message passing GNNs in general.

By further investigating the ideas developed in this chapter, stronger theoretical guarantees for GNN weak recovery performance may be obtained. Such results will strengthen the theory of GNNs, which in turn will lead to improvements in architectures, training protocols, and implementation for graph learning algorithms.

Chapter 4. Improving a Result on Size Generalization of GNNs

In this short chapter we improve upon a result in [35] by improving upon an underlying theorem from [36]. The result deals with the well-documented size generalization problem for GNNs. GNNs trained on small datasets tend to underperform when used on large datasets [35]. [35] suggest this occurs because neighborhood degree patterns tend to differ on larger graphs in comparison to smaller graphs. In this paper, Yehudai et al demonstrate that for any task solvable by a depth d GNN, there exissts some GNN which simultaneously performs well on the training data and poorly on data pulled from another distribution, with the performance gap depending on how much the degree patterns of the two distributions differ.

Their result rests on the ability to obtain a 3 layer ReLU feedforward neural network that can "shatter" the data, which is obtained in [36]. We noticed that this result can be extended to 2 layer ReLU feedforward neural networks with just a little more work. We restate the theorem from [36] as follows:

Theorem 4.1 (Corollary 4.2 from [36])**.** *Consider any dataset $\{(x_i, y_i)\}_{i=1}^N$ that satisfies the assumption that the x_i are in general position. Suppose 2-layer ReLU-like feedforward neural network f_θ satisfies*

$$d_i \geq \frac{4N}{d_x} + 4d_y$$

Then there exists a parameter set θ such that $y_i = f_\theta(x_i)$ for all i.

Note that this result depends on the x_i being in general position. We will now argue that that assumption is not necessary.

If the x_i are in general position, the result applies. If not, then in [37] Munkres provides a theorem allowing us to obtain a perturbed dataset $\{\tilde{x_i}\}_{i=1}^N$ in general position for which $\|x_i - \tilde{x_i}\| < \delta$ for a chosen $\delta > 0$ and all i. Since we can do this, we would like to demonstrate the result for arbitrary x_i as the limit of perturbed datasets in general position.

For each $n \in \mathbb{N}$, choose a perturbed dataset $\{x_{n,i}\}_{i=1}^N$ in general position such that $\|x_{n,i} - x_i\| < \frac{1}{n}$ for all i. Then using the general position assumption, apply the theorem above to obtain a parameter set θ_n for each n such that $f_{\theta_n}(x_{n,i}) = y_i$ for all i. Then we have $\lim_{n\to\infty}(x_{n,i}) = x_i$, but it is not immediately clear whether the θ_n should converge or not. It is sufficient to show that the θ_n are bounded to obtain a subsequence θ_{n_j} converging to a particular θ. If this is the case then we may obtain through continuity of f as a function of the parameters as well as the input data

$$y_i = \lim_{j\to\infty}(y_i) = \lim_{j\to\infty}(f_{\theta_{n_j}}(x_{n_j,i})) = f_\theta(x_i)$$

Which demonstrates that θ satisfies the conclusion of the theorem.

We now show that the θ_n can be chosen to be bounded. To do this we heavily reference the proof of Theorem 4.1 (and hence corollary 4.2) in [36]. We only need to worry about bounding a handful of parameters for each n. We start with two "incoming" parameters for layer l and hidden embedding j:

$$U^l_{j,:} = \alpha u^T, \quad b^l_j = \alpha c,$$

where u is a vector (of arbitrary magnitude) for a hyperplane equation $u^T x + c = 0$ obtained through the general position assumption isolating a selection of the $x_{n,i}$ relevant to the proof. Then α is chosen to make $|\alpha(u^T x_{n,i} + c)| > 1$ for all i. Because the datasets $\{x_{n,i}\}_{i=1}^N$ converge to the dataset $\{x_i\}_{i=1}^N$ and each dataset contains finitely many points the datasets may be absolutely bounded in size. Thus α can be chosen such that $U^l_{j,:}$ and b^l_j are absolutely bounded across all the parameter sets θ_n by some arbitrarily large constant not depending on n. Next we worry about the "outgoing" parameters

$$V^l_{:,j} = \beta e_k, \quad c^l = 0$$

Where e_k is a standard component vector (therefore of size 1) and β is chosen so that the k-th component of the hidden embedding $h^l(x_i)$ is larger than a value which is bounded by some constant regardless of n. We also choose β not to break the general position assumption, but as noted in the proof given in [36], those points lie in a set of measure zero. So β can be chosen to be bounded above as well. All other parameters are bounded absolutely in the original proof, so we may bound all of the θ_n. Hence by the argument above we obtain Theorem 4.1 in the general case where the data $\{x_i\}_{i=1}^N$ are not required to be in general position. Thus we may improve the bound in Theorem 5.2 of [35] from $d+2$ to $d+1$ applying the same method of construction as Yehudai et al do in their proof using the 2-layer GNN obtained in this new result.

Appendix A. Additional SBM Experiments

A.1 Means

The comprehensive results for mean values of our experiments are found below. Additional experiments using other architectures or wider bounds may be conducted using our code in GitHub https://github.com/brownthesr/Synthetic-Graphs.git.

When considering the Poisson SBM, we can see that SAGE performed the best of any GNN architecture across any class size. It should also be noted that as we increase the number of classes more information is needed for any architecture to correctly classify. Additionally, the heterophilous regime is more adversely affected by the increase in class size than is the homophilous regime. By comparing Figure A.1 and Figure A.2 we can observe how each model is affected by degree correction across any number of classes.

In Figure A.3 and Figure A.4 we view the various regimes across which each architecture outperforms the others. As we increase the class sizes, the favorable regime for the neural network increases in size, showing that in many cases it is simply better to ignore edges and utilize solely the feature information. However, it should be noted that in most of the cases, there is always a regime where the GNN architecture outperforms both of the baselines.

Poisson Degree Distribution

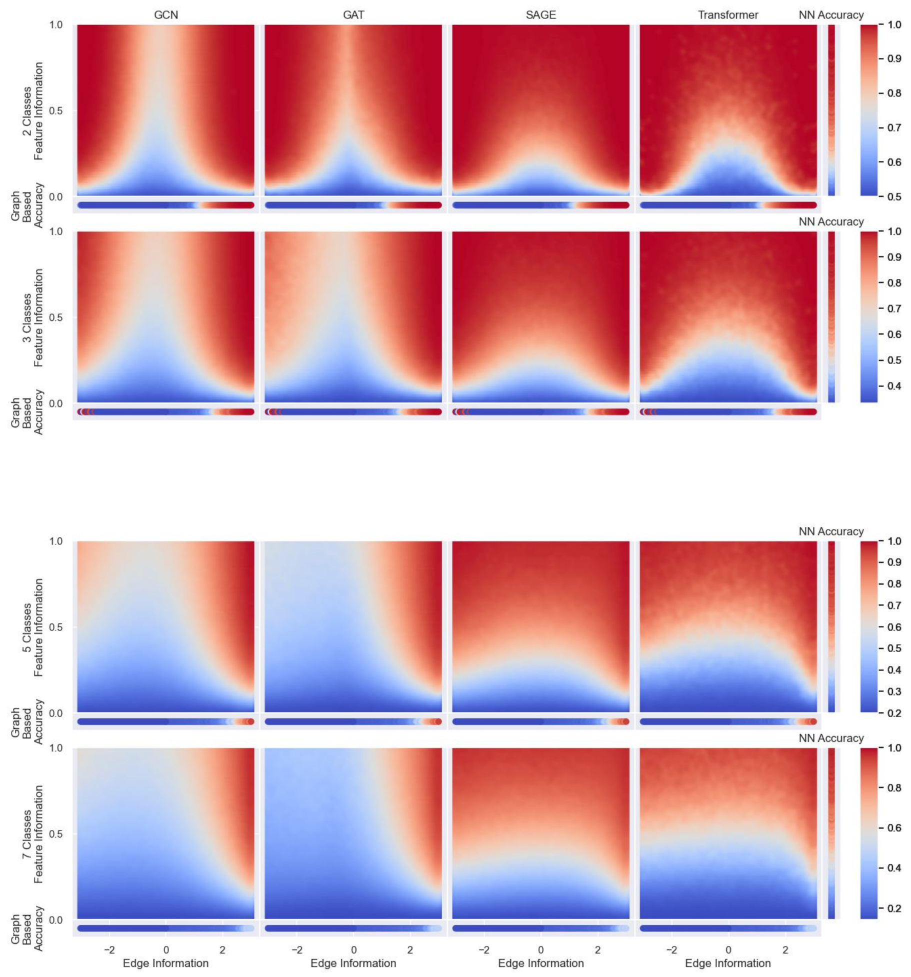

Figure A.1: We compare accuracies over the distributed Poisson graphs across varying class sizes. We also depict the accuracy curves of a regular feedforward neural network and that of spectral clustering on the same datasets to the right and bottom of each plot respectively. Using these plots we can compare how well each architecture performs on an increased number of classes. Additionally we can view how performance changes across different architectures.

Degree-Corrected Degree Distribution

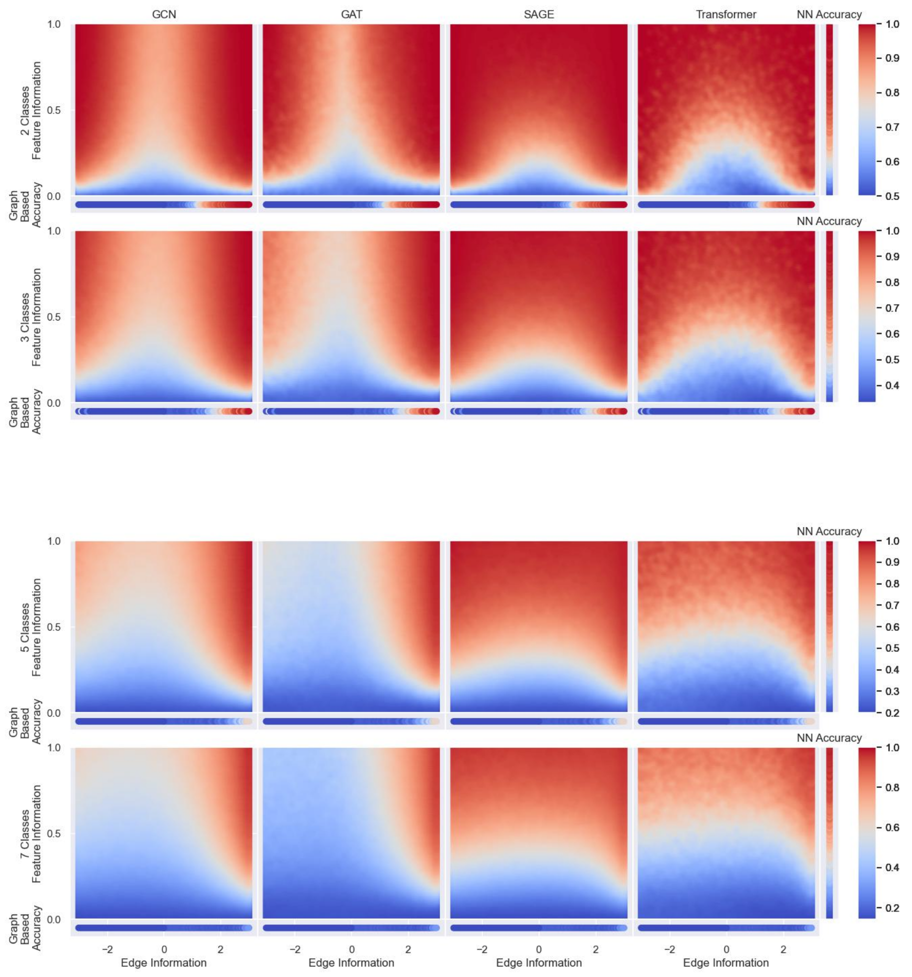

Figure A.2: We compare accuracies over degree-corrected graphs across varying class sizes. Across any number of classes, the GCN did better on degree corrected graphs. This can be seen by viewing how the blue region in the top figures shrinks in the degree-corrected case. The performance of the Transformer improved in degree-corrected cases for class numbers of two and three, yet it decreased performance for class numbers of five and seven. The performance of SAGE and GAT were mostly unaffected by the degree correction.

Poisson Degree Distribution

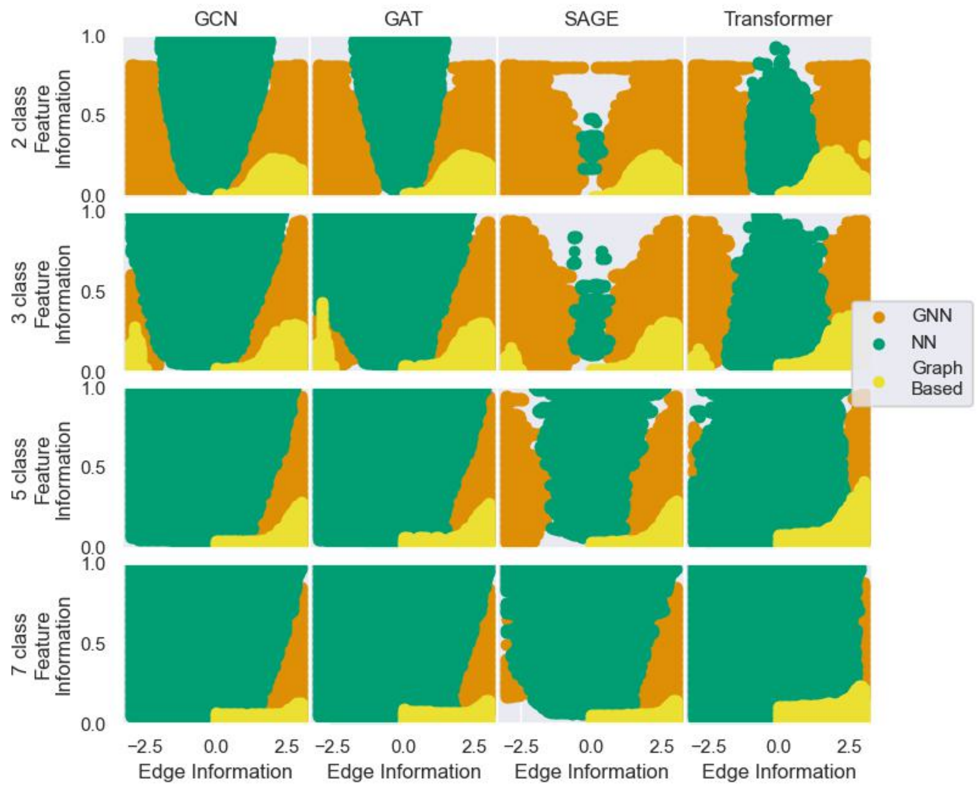

Figure A.3: Regions of where each architecture outperforms the others across the Poisson SBM graphs. Here we can compare how varying parts of the data effects the shape and sizes of the favorable regimes

Degree-Corrected Degree Distribution

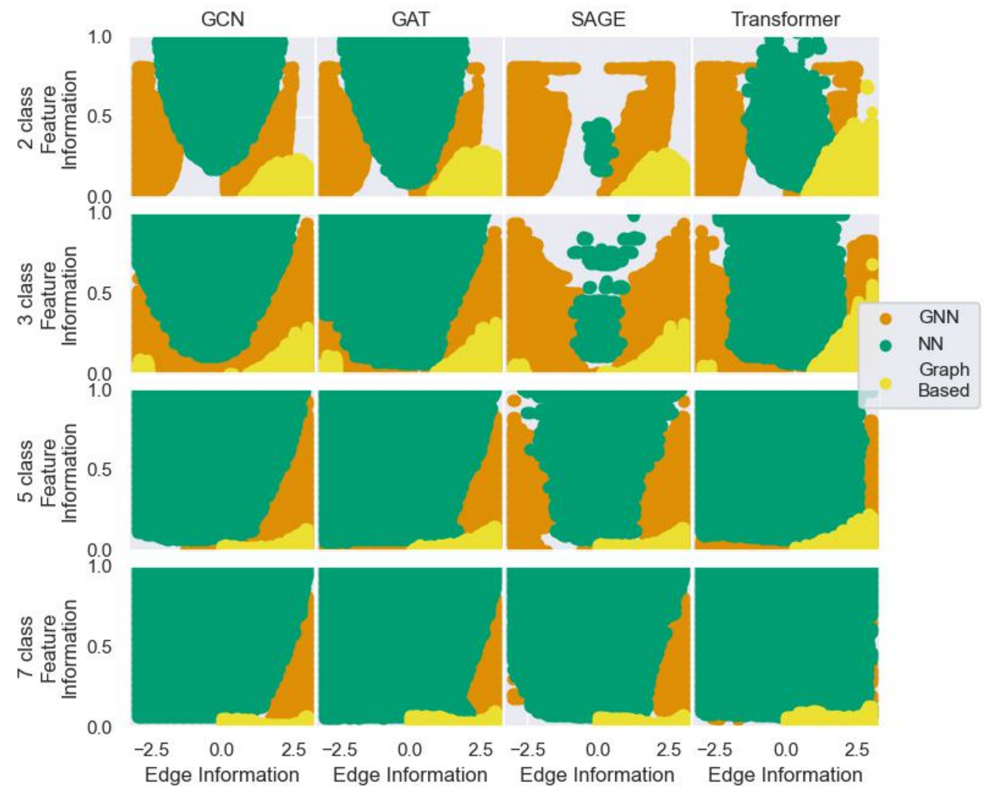

Figure A.4: Regions where each architecture outperforms the others across degree-corrected graphs.

A.2 Maxes

The comprehensive results for max values of our experiments are found below. Additional experiments using other architectures or wider bounds may be conducted using our code in

Poisson Degree Distribution

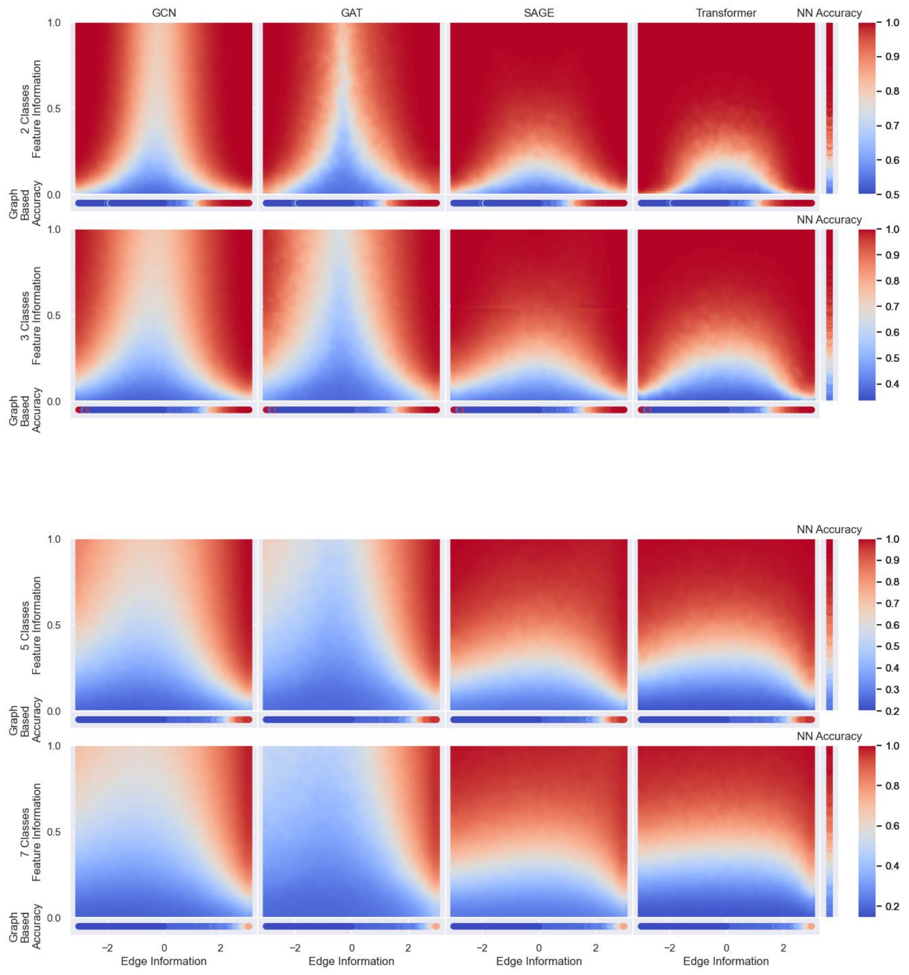

Figure A.5: We compare accuracies over the distributed Poisson graphs across varying class sizes. Note that the blue regions of these graphs are much more pointed than the mean graphs

When we view the maxes in light of the average graphs, we see that the blue portions of the maxes are much more steeply shaped than that of the averaged. This likely demonstrates that while both the averages and the maxes perform poorly towards the middle (where there is a lot of edge noise) the model is able to achieve better along the sides of the graph where we have less feature information but more edge information. In general it seems that the models

Degree-Corrected Degree Distribution

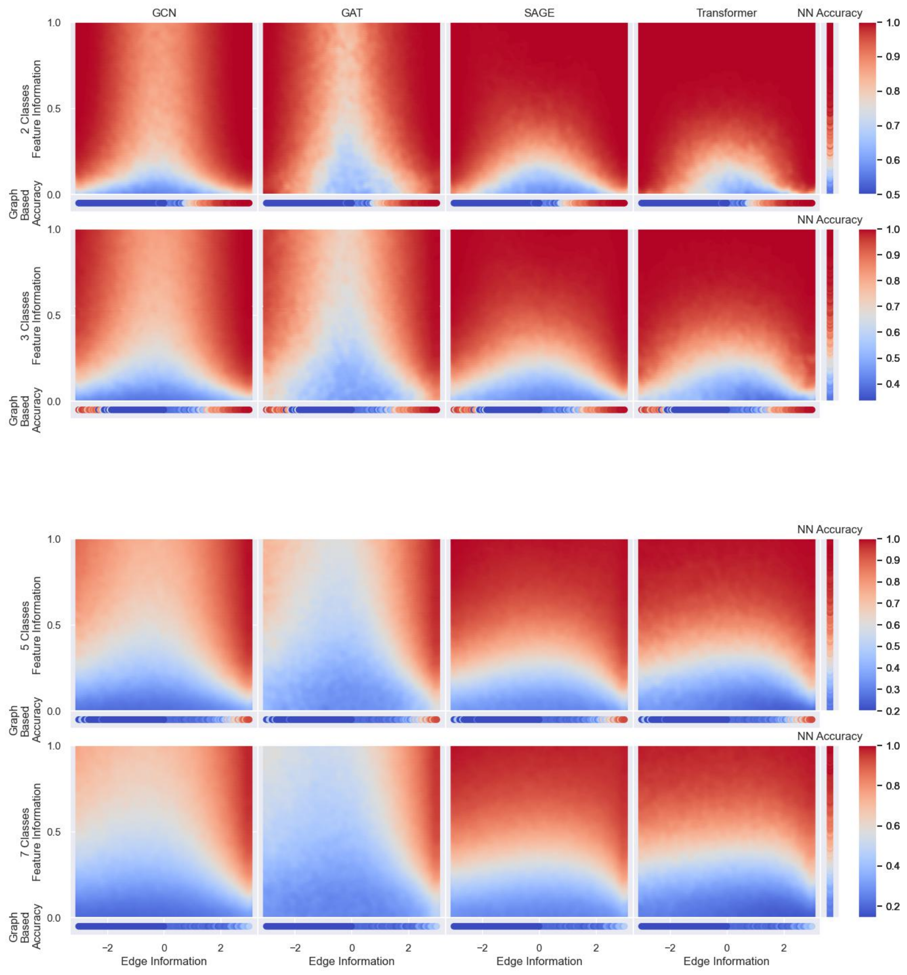

Figure A.6: We compare accuracies over degree-corrected graphs across varying class sizes.

benefited from operating on heavy-tailed graphs. In particular we see that the GCN and the GAT performed better on degree corrected graphs across all class sizes. The Transformer and SAGE did not see as stark an increase in performance, but did perform noticeably better on class sizes of 2 and 3.

Poisson Degree Distribution

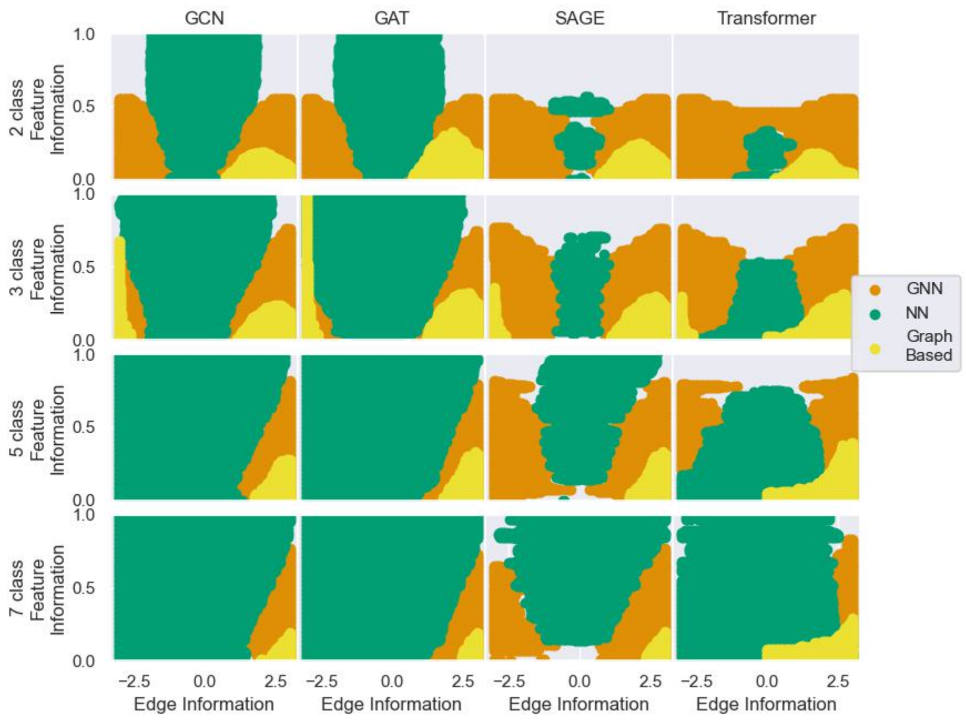

Figure A.7: Regions of where each architecture outperforms the others across the Poisson SBM graphs. Here we can compare how varying parts of the data effects the shape and sizes of the favorable regimes

Degree-Corrected Degree Distribution

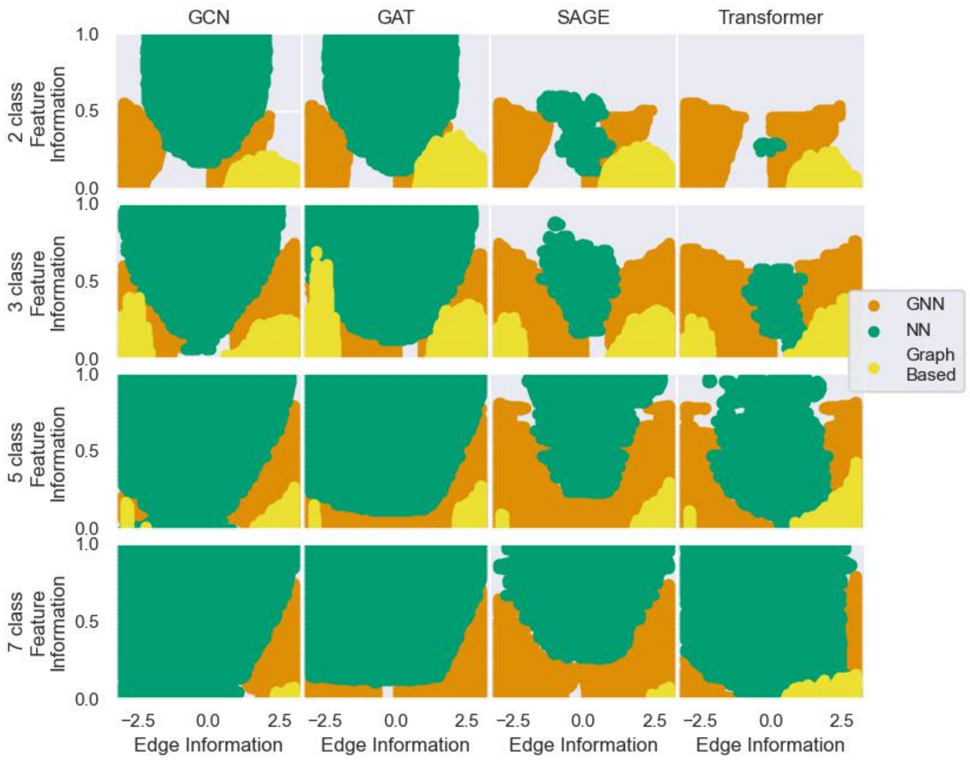

Figure A.8: Regions where each architecture outperforms the others across degree-corrected graphs.

Appendix B. Proof of Single Layer GCN Probability Result

Let

$$\Sigma = \begin{pmatrix} \sigma_x & \sigma_{xy} \\ \sigma_{xy} & \sigma_y \end{pmatrix}$$

Be the invertible covariance matrix and denote $|\Sigma| = \det \Sigma$. We begin with the expression

$$\int_{-\infty}^{\delta} \int_{-\infty}^{\infty} \frac{1}{2\pi\sqrt{|\Sigma|}} \exp\left(-\frac{1}{2}(x,y)\Sigma^{-1}(x,y)^T\right) dx\, dy$$

We expand and simplify the matrix multiplication to obtain

$$= \int_{-\infty}^{\delta} \int_{-\infty}^{\infty} \frac{1}{2\pi\sqrt{|\Sigma|}} \exp\left(-\frac{\sigma_y}{2|\Sigma|}x^2 + \frac{\sigma_{xy}}{|\Sigma|}xy + \frac{\sigma_x}{2|\Sigma|}y^2\right) dx\, dy$$

$$= \frac{1}{2\pi\sqrt{|\Sigma|}} \int_{-\infty}^{\delta} \int_{-\infty}^{\infty} \exp\left(-\frac{\sigma_y}{2|\Sigma|}x^2 + \frac{\sigma_{xy}}{|\Sigma|}xy + \frac{\sigma_x}{2|\Sigma|}y^2\right) dx\, dy$$

We then notice that the inside integral is just a Gaussian integral from $-\infty$ to ∞, and apply a known formula.

$$= \frac{1}{2\pi\sqrt{|\Sigma|}} \int_{-\infty}^{\delta} \sqrt{\frac{\pi}{\left(\frac{\sigma_y}{2|\Sigma|}\right)}} \exp\left(\frac{\left(\frac{\sigma_{xy}}{|\Sigma|}y\right)^2}{4\left(\frac{\sigma_y}{2|\Sigma|}\right)} - \frac{\sigma_x}{2|\Sigma|}y^2\right) dy$$

Simplifying, we obtain:

$$= \frac{1}{\sqrt{2\pi\sigma_y}} \int_{-\infty}^{\delta} \exp\left(\frac{\sigma_{xy}^2 - \sigma_x\sigma_y}{2\sigma_y|\Sigma|}y^2\right) dy$$

Noting that $\sigma_{xy}^2 - \sigma_x\sigma_y = -|\Sigma|$ we then have

$$= \int_{-\infty}^{\delta} \frac{1}{\sqrt{2\pi\sigma_y}} \exp\left(-\frac{1}{2}\left(\frac{y}{\sqrt{\sigma_y}}\right)^2\right) dy$$

Which is the cumulative distribution function of a Gaussian normal distribution with mean 0 and variance σ_y. Hence we obtain:

$$= \frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{\delta}{\sqrt{2\sigma_y}}\right)\right)$$

Where erf is the Gaussian error function. □

Bibliography

[1] V.A. Traag, Waltman. L., and N.-J. Van Eck. leidenalg documentation — leidenalg 0.9.1.dev0+g46cea68.d20221003 documentation, 2016.

[2] Emmanuel Abbe. Community detection and stochastic block models. *Proceedings in Machine Learning Research*, 2017.

[3] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *Internation Conference for Learning Representations*, 2019.

[4] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in Neural Information Processing*, 2020.

[5] Sitao Luan, Chenqing Hua, Qincheng Lu, Jiaqi Zhu, Mingde Zhao, Shuyuan Zhang, Xiao-Wen Chang, and Doina Precup. Is heterophily a real nightmare for graph neural networks to do node classification?, 2021. arXiv:2109.05641[cs.].

[6] Kimon Fountoulakis, Amit Levi, Shenghao Yang, Aseem Baranwal, and Aukosh Jagannath. Graph attention retrospective, 2022. arXiv:2202.13060.

[7] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. *Internation Conference for Learning Representations*, 2018.

[8] Aseem Baranwal, Kimon Fountoulakis, and Aukosh Jagannath. Effects of graph convolutions in multi-layer networks, 2022. arXiv:2204.09297.

[9] Hoang NT and Takanori Maehara. Revisiting graph neural networks: All we have is low-pass filters, May 2019. arXiv:1905.09550 [cs, math, stat].

[10] Brian Karrer and M. E. J. Newman. Stochastic blockmodels and community structure in networks. *Pubmed*, 2011.

[11] Chao Gao, Zongming Ma, Anderson Y. Zhang, and Harrison H. Zhou. Community detection in degree-corrected block models. *The Annals of Statistics*, 46(5), October 2018.

[12] Yash Deshpande, Andrea Montanari, Elchanan Mossel, and Subhabrata Sen. Contextual stochastic block models. *Advances in Neural Information Processing*, 2018.

[13] Danny Bickson. Gaussian belief propagation: Theory and application, July 2009. arXiv:0811.2518 [cs, math].

[14] Norbert Binkiewicz, Joshua T. Vogelstein, and Karl Rohe. Covariate-assisted spectral clustering. *Biometrika*, 104(2):361–377, June 2017.

[15] Liang Yang, Wenmiao Zhou, Weihang Peng, Bingxin Niu, Junhua Gu, Chuan Wang, Xiaochun Cao, and Dongxiao He. Graph neural networks beyond compromise between attribute and topology. In *Proceedings of the ACM Web Conference 2022*, pages 1127–1135. ACM, April 2022.

[16] Jesús Arroyo, Avanti Athreya, Joshua Cape, Guodong Chen, Carey E. Priebe, and Joshua T. Vogelstein. Inference for multiple heterogeneous networks with a common invariant subspace, August 2020. arXiv:1906.10026 [cs, math, stat].

[17] Pan Li and Jure Leskovec. The expressive power of graph neural networks. In Lingfei Wu, Peng Cui, Jian Pei, and Liang Zhao, editors, *Graph Neural Networks: Foundations, Frontiers, and Applications*, pages 63–98. Springer, Singapore, 2022.

[18] Ningyuan Huang and Soledad Villar. A short tutorial on the Weisfeiler-Lehman test and its variants. In *2021 - 2021 IEEE International Conference on Acoustics*, pages 8533–8537, June 2021.

[19] William L. Hamilton. Theoretical motivations. In *Graph Representation Learning*, pages 77–103. Springer, Cham, 2020.

[20] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *Internation Conference on Learning Representations*, 2017.

[21] Dexiong Chen, Leslie O'Bray, and Karsten Borgwardt. Structure-aware transformer for graph representation learning. *International Conference for Machine Learning*, 2022.

[22] Ulrike von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 2007.

[23] Judea Pearl. Reverend Bayes on inference engines: A distributed hierarchical approach. In Hector Geffner, Rina Dechter, and Joseph Y. Halpern, editors, *Probabilistic and Causal Inference*, pages 129–138. ACM, NY, USA, February 2022.

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, December 2017. arXiv:1706.03762 [cs].

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[26] Yao Ma, Xiaorui Liu, Neil Shah, and Jiliang Tang. Is homophily a necessity for graph neural networks?, 2022. arXiv:2106.06134.

[27] Réka Albert, Hawoong Jeong, and Albert-László Barabási. Error and attack tolerance of complex networks. *Nature*, 406(6794):378–382, July 2000.

[28] Bailey K. Fosdick, Daniel B. Larremore, Joel Nishimura, and Johan Ugander. Configuring random graph models with fixed degree sequences, 2016.

[29] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. GraphSAINT: Graph sampling based inductive learning method, February 2020. arXiv:1907.04931 [cs, stat].

[30] Benedek Rozemberczki and Rik Sarkar. Characteristic functions on graphs: Birds of a feather, from statistical descriptors to parametric models, August 2020. arXiv:2005.07959 [cs, stat].

[31] Aleksandar Bojchevski and Stephan Günnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking, February 2018. arXiv:1707.03815 [cs, stat].

[32] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding, March 2021. arXiv:1909.13021 [cs, stat].

[33] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation, June 2019. arXiv:1811.05868 [cs, stat].

[34] Wei Lu. Learning guarantees for graph convolutional networks in the stochastic block model, 2022.

[35] Gilad Yehudai, Ethan Fetaya, Eli Meirom, Gal Chechik, and Haggai Maron. From local structures to size generalization in graph neural networks, 2021.

[36] Chulhee Yun, Suvrit Sra, and Ali Jadbabaie. Small relu networks are powerful memorizers: a tight analysis of memorization capacity, 2019.

[37] J.R. Munkres. *Topology: A First Course*. Prentice Hall International, 1997.