University College London

Master's Thesis
MSc in Machine Learning

On the Effects of DropEdge on Over-squashing in GNNs

Author:
Jasraj Singh

Supervisors:
Laura Toni
Brooks Paige

November 3, 2024

To the friend I lost, I have missed you.

Acknowledgements

I would like to express my sincerest gratitude towards my supervisors, Professor Laura Toni and Professor Brooks Paige, for their invaluable guidance, support, and encouragement throughout the course of this research. Their patience and guidance helped me grow both as a researcher and as an individual. I would also like to express my heartfelt thanks to Keyue Jiang for his mentorship and companionship throughout this project.

To my family, I cannot thank you enough for the opportunities you have provided. And to my friends, thank you for your unwavering support throughout this journey. I have always felt at home with you guys.

Abstract

Message Passing Neural Networks (`MPNNs`) are a class of Graph Neural Networks (`GNNs`) that leverage graph topology to propagate messages across increasingly larger neighborhoods. The message-passing scheme leads to two distinct challenges: *over-smoothing* and *over-squashing*. While several algorithms have successfully addressed the over-smoothing issue, their impact on over-squashing remains largely unexplored. This represents a critical gap in the literature, as failing to mitigate over-squashing implies that these methods may not be suitable for long-range tasks. In this thesis, we take the first step toward closing this gap by studying the `DropEdge` algorithm in the context of over-squashing. We present novel theoretical findings that characterize its detrimental effects on sensitivity between distant nodes, suggesting its unsuitability for long-range tasks. We further evaluate `DropEdge` models on synthetic and real-world datasets, demonstrating its negative effects. Additionally, we propose a new hypothesis about `DropEdge`'s influence on `GNNs` – while it effectively increases model-dataset alignment on short-range tasks, it misaligns them on long-range tasks, leading to *high approximation error* and *overfitting* to short-range artifacts in the training data. Our work emphasizes the need to evaluate methods designed for training deep `GNNs`, with a renewed focus on long-range graph benchmarks.

The codebase associated to this thesis is available at https://github.com/ignasa007/DropEdge-on-OverSquashing.

Contents

Chapter 1

Introduction

1.1 Overview

In the modern age, graph-structured data is ubiquitous. It is found in various domains like social media platforms, online retail platforms, molecular structures, transportation networks, and even computer systems. Understanding and modeling these graphs has become increasingly important, attracting significant attention from a broad range of adjacent fields. This includes the rapidly growing area of machine learning[1], driven by the promise of deep learning algorithms that have profoundly impacted our daily lives. Numerous algorithms, including *graph neural networks* (`GNNs`) [Sca+09; Li+16], have been developed for learning on graphs, supported by a wide array of engineering tools, like `PyTorch Geometric` [FL19], `Tensorflow GNN` [Fer+23] and the `Deep Graph Library` [Wan+19], which have enabled rapid development. `GNNs` have proven to be powerful tools for modelling graph-structured data, and have found applications in real-world scenarios like inferring relations in social networks [GWJ18; You+20a; You+20b; You+20c; You+22], improving recommendation systems [MBB17; Yin+18; Zhe+22], and molecular modelling for biochemical applications [WK06; ZL17], amongst others. Owing to their impressive performance, `GNNs` have become a popular choice for graph analysis in recent times.

[1]https://x.com/prlz77/status/1178662575900368903

Drawing inspiration from message-passing algorithms used for inference in probabilistic graphical models [YFW03], a popular class of `GNNs`, called *message-passing neural networks* (`MPNNs`) [Gil+17], recursively process node-level and edge-level features using message-passing layers. These layers are stacked to allow each node to aggregate information from an increasingly larger neighborhood, akin to how *convolutional neural networks* (`CNNs`) learn hierarchical features for images [LeC+89]. However, unlike in image-based deep learning, where *ultra-deep* `CNN` architectures have led to performance breakthroughs [Sze+15, 22 layers; He+16, 152 layers], shallow `GNNs` often outperform deeper models on many graph learning tasks [Zho+21b]. In some cases, like with citation networks [Sen+08; YCS16] and product recommendations [McA+15; Shc+18], the learning tasks are inherently *short-range*, meaning that local information-aggregation is sufficient for approximating the ground-truth. Hence, deeper networks can result in model misspecification, leading to poor generalization to unseen data. In other cases, we have tasks requiring *long-range* interactions (LRIs) for making good predictions. These include point clouds [Li+19], meshes [Gon+20], and molecular networks whose chemical properties may be determined by diametrically opposite atoms [MDS18]. Unfortunately, simply stacking more message-passing layers does not suffice. Rather, it can be detrimental to model performance. This is because deep `GNNs` suffer from unique issues like *over-smoothing* [LHW18; OS20] and *over-squashing* [AY21; Top+22], which makes training them notoriously difficult.

Over-smoothing refers to the problem of layer-wise convergence of some node-similarity measure [CW20a; Che+20a], i.e. node representations become *too similar* as they are recursively processed using more and more layers [LHW18; OS20]. This is undesirable since this limits the `GNN` output to depend only on the graph topology, failing to utilize the information in the node features. Over-smoothing has garnered significant attention from the research community, leading to both, theoretical results studying the phenomenon and algorithms designed specifically to address it (see [RBM23] for a survey). Amongst these methods is `DropEdge` [Ron+20], an implicit regularization technique which adds noise to the optimization process by randomly dropping edges from the graph. Although `DropEdge` was, theoretically and empirically, shown to reduce over-smoothing, the performance metrics on real-world datasets progressively worsen as more layers are stacked [Ron+20; Zho+21b]. This implies that there continue to be issues that `DropEdge` does not resolve. *Or, perhaps, DropEdge introduces new problems which have gone unnoticed up till now.*

The other issue specific to GNNs is over-squashing. When the task entails modelling LRIs, information needs to be propagated across the graph without significant loss. However, for certain graph structures, e.g. "small world" graphs like social networks, the neighborhood size grows exponentially as the distance from a node grows [CZS18]. In such cases, information may be lost as it is *squashed* through graph *bottlenecks* [AY21]. This makes it harder for MPNNs to facilitate communication between nodes at long distances, which is imperative for good performance on long-range tasks. In other words, over-squashing is undesirable since it reduces the *effective receptive field* of the models. To alleviate over-squashing, several graph-rewiring techniques have been proposed [AY21; DLV22; Bla+23; KBM23; Ngu+23] that, in some sense, aim to improve the connectivity of the graph. With all these methods, one common idea is that edges need to be added in a strategic manner to alleviate over-squashing. Interestingly, DropEdge [Ron+20], which is one of the more popular methods for training deep GNNs, only removes edges. This should, in principle, amplify over-squashing levels, making it harder for models to capture LRIs.

The empirical evidence in support of DropEdge, as well as most other methods designed for training deeper GNNs, has been mainly collected on short-range tasks. That is, it simply suggests that *these methods prevent loss of local information, but whether they facilitate capturing LRIs remains inconclusive*. Of course, on long-range tasks, deeper GNNs are useless if they cannot capture such interactions. This concern is also relevant for other methods aimed at alleviating over-smoothing, especially since evidence suggests that this could result in a trade-off with over-squashing [Gir+23; Ngu+23]. Unfortunately, possibly because over-squashing is a more recently discovered phenomenon, there has been little research into how such methods affect it. However, with our growing understanding of over-squashing and its negative impacts, it is essential to revisit these ideas and determine their applicability to long-range tasks.

1.2 Research Objectives

Given the concerns raised about the effectiveness of existing methods for training deeper GNNs on long-range tasks, it is crucial to re-evaluate their capability in this context. This re-evaluation is particularly important as methods like DropEdge have shown success in short-range tasks but remain understudied in their ability to mitigate over-squashing and model LRIs. In this work, we aim to uncover the effects of `DropEdge` on over-squashing in `MPNNs`, and to gain a better understanding of whether `MPNNs` benefit from its use on long-range learning tasks, like they do on short-range ones. We will pursue this objective in two ways: 1. *indirectly*, by studying the effect of `DropEdge` on the expected commute time in the computational graph, and 2. *directly*, by studying its effects on the sensitivity of node representations [Top+22], as well as on performance on synthetic and real-world tasks. By achieving these objectives, this study will offer critical insights into the role of `DropEdge` in the context of the over-squashing phenomenon and its effect on the long-range task performance of `MPNNs`. These findings will also help inform the design and evaluation of future `GNN` architectures, ensuring that models can handle both short-range and long-range dependencies effectively.

1.3 Contributions

The main contribution of this thesis is a theoretical characterization of the effect of `DropEdge` on over-squashing. Specifically, by computing the expected sensitivity of the node representations (inversely related to over-squashing) in a linear `GCN`, we show that `DropEdge` provably reduces the effective receptive field of the model. Furthermore, we extend the existing theoretical results on sensitivity in non-linear `MPNNs` [Xu+18; Bla+23; Di +23] to the `DropEdge` setting, again showing that it exacerbates the over-squashing problem. Finally, we show that all our conclusions about the training-time setting of `DropEdge` continue to hold for its test-time setting [XZL23].

We also provide experimental support for these results: firstly, in a training-free setting, we show that `DropEdge` reduces the average distance to which information from a source propagates. Next, with a semi-synthetic dataset [Gio+24], we show that turning `DropEdge` on leads to a significant decline in performance on long-range tasks. Finally, we show that as the `DropEdge` probability increases, the performances of `GNNs` exhibit starkly contrasting trends between citation networks (`Cora` [McC+00] and `CiteSeer` [GBL98]) and molecular datasets (`Proteins` [DD03] and `MUTAG` [KMB05]). Accordingly, we propose a novel hypothesis on the effect of `DropEdge`: for short-range tasks, `DropEdge` increases model-dataset alignment by reducing the receptive field of the `GNN`, thereby enhancing its performance. Conversely, for long-range tasks, it decreases the model-dataset alignment, resulting in poorer performance.

Our results suggest a need for a re-evaluation of methods designed to train deep `GNNs`, with a focus on their performance on long-range tasks. Indeed, trainability of deep `GNNs` does not necessarily imply that models can effectively propagate information over long distances [AY21]. Especially with the reported trade-off between over-smoothing and over-squashing [Gir+23; Ngu+23], there is a good reason to be skeptical about the suitability of these methods towards modelling LRIs. Our work thoroughly examines `DropEdge` from both theoretical and empirical perspectives; however, there are still many other techniques that require further analysis.

1.4 Outline

The rest of this thesis is organised as follows. In Chapter 2, we will present the literature relevant to our work. We will start with basic graph-theoretic concepts (Section 2.1) and then move to introducing `MPNNs` (Section 2.2), `DropEdge` (Section 2.3), and over-squashing (Section 2.4). We will conclude this section by discussing some related works (Section 2.5) – these are not necessary for understanding our results, but are there to provide additional context for it. In Chapter 3, we will present our novel theoretical results. We will start by studying what we call a `DropEdge` *random walk* (Section 3.1), showing that its commute times are larger than in a regular random walk. From there, we will move on to studying the effect of `DropEdge` on over-squashing (Section 3.2) – we will start with simple models, and progressively consider more general classes of `MPNNs`. In Chapter 4, we provide empirical evidence supporting our theoretical results. Section 4.1 presents the inverse relationship between *propagation distance* and commute time, as well as the negative effect of `DropEdge` on it. In Section 4.2, we evaluate models on the `SyntheticZINC` dataset [Gio+24], again concluding that `DropEdge` hurts model performance. Finally, in Section 4.3, we evaluate the `GCN` and `GAT` models on real-world datasets, noting stark differences in the trends in accuracy as the `DropEdge` probability is increased. Accordingly, we present a novel hypothesis relating the effect of `DropEdge` on performance to the *problem-radius*.

1.5 A Note on Notation

Throughout the thesis, we will try our best to maintain consistent notations. Although we will introduce new notations as we proceed, for most part we will use the following LaTeX fonts to denote different mathematical objects:

Object	**Font**	**Case**	**Examples**
Scalars	`\mathnormal`	Lower or Upper	a, A, b, B
Vectors	`\mathbf`	Lower	$\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$
Matrices	`\mathbf`	Upper	$\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$
Number Sets	`\mathbb`	Upper	$\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$
Other Sets	`\mathcal`	Upper	$\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$
Functions	`\mathnormal` or `\mathsf`	Lower and Upper	d, f, GNN, ReLU

Table 1.1: Font Conventions

Chapter 2

Background

This chapter is dedicated to providing the reader the necessary context for our work. We will start with some graph theoretic foundations in Section 2.1, presenting the related notational convention for this thesis (Section 2.1.1) and a discussion on 3 topological properties: Cheeger's constant (Section 2.1.2), effective resistance (Section 2.1.3) and expected commute time (Section 2.1.4). In the following sections, we will get into more `GNN` specific literature. In Section 2.2, we will introduce `MPNNs`. We advise the reader to pay close attention to the details of the `GCN` architecture in Section 2.2.1, since this will be our choice of model to study in Chapter 3. We will also introduce the `GAT` model (Section 2.2.2), which will be evaluated alongside `GCN` in Section 4.3. In Section 2.3, we will introduce the `DropEdge` algorithm, and in Section 2.4, we will introduce theoretical results characterizing the over-squashing phenomenon. Specifically, in Section 2.4.1, we will introduce sensitivity of node representations as a measure of over-squashing, and relate it to the graph topology. In Section 2.4.2, we will use sensitivity to define the influence distribution, and draw its equivalence with the augmented random walk on the graph. Finally, in Section 2.4.3, we will introduce the Jacobian obstruction as another measure of over-squashing, and characterize it using the effective resistance between nodes. To conclude this chapter, we will present some related works in Section 2.5 which are relevant for understanding the significance of our contributions. This includes algorithms for addressing over-smoothing (Section 2.5.1), over-squashing (Section 2.5.2) and their trade-off (Section 2.5.3), as well as long-range graph benchmarks (Section 2.5.4).

2.1 Introduction to Graph Theory

[Top+22; Bla+23; Di +23; Gio+24] have shown that over-squashing in a `GNN` is intricately related to topological properties of the graph, such as the *curvature* of its edges and the *effective resistance* between two nodes. Accordingly, we will introduce 3 key topological properties in this section: *Cheeger's constant*, effective resistance and *expected commute time*. Understanding these properties is crucial not only for interpreting the existing literature on over-squashing but also for motivating our approach of studying the impact of `DropEdge` on over-squashing via the commute time in its computational graph. These insights will allow us to discuss quantitative measures of over-squashing and explore the theoretical results that link these measures to the topological properties. To facilitate this discussion, we will start by setting up some basic notation.

2.1.1 Notation

Consider a weighted directed graph $\mathsf{G}(\mathcal{V}, \mathcal{E}, W)$, where $\mathcal{V} = [N] \coloneqq \{1, \ldots, N\}$ is the node set, $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ is the edge set, where $(i, j) \in \mathcal{E}$ if there's an edge from node i to node j, and $W : \mathcal{E} \to \mathbb{R} \setminus \{0\}$ maps each edge to an edge weight. In this work, we assume that the edge weights are positive, i.e. $W : \mathcal{E} \to \mathbb{R}_{>0}$.

Definition 2.1 (Undirected Graph). *If a graph $\mathsf{G}(\mathcal{V}, \mathcal{E}, W)$ is undirected, then $(i, j) \in \mathcal{E}$ if and only if $(j, i) \in \mathcal{E}$. Moreover, the edge weights are symmetric, i.e. $W(i, j) = W(j, i)$.*

Definition 2.2 (Unweighted Graph). *If a graph $\mathsf{G}(\mathcal{V}, \mathcal{E}, W)$ is unweighted, then $W(i, j) = 1, \forall (i, j) \in \mathcal{E}$. We denote unweighted graphs by $\mathsf{G}(\mathcal{V}, \mathcal{E})$.*

Definition 2.3 (Adjacency Matrix). *The adjacency matrix, $\mathbf{A} \in \mathbb{R}_{\geq 0}^{N \times N}$, of the graph $\mathsf{G}(\mathcal{V}, \mathcal{E}, W)$ is given as*

$$\mathbf{A}_{ji} \coloneqq \begin{cases} W(i, j), & (i, j) \in \mathcal{E} \\ 0, & (i, j) \notin \mathcal{E} \end{cases}$$

Note that the adjacency matrix of an undirected graph is symmetric, i.e. $\mathbf{A} = \mathbf{A}^T$, and that of an unweighted graph is a binary matrix, i.e. $\mathbf{A} \in \{0, 1\}^{N \times N}$.

Lemma 2.1. *In a graph $\mathsf{G}(\mathcal{V},\mathcal{E})$, the powers of the adjacency matrix encode the number of paths of different lengths, i.e. for $l \in \mathbb{N}$, $\left(\mathbf{A}^l\right)_{ji}$ is the number of paths of length l starting at node $i \in \mathcal{V}$ and ending at node $j \in \mathcal{V}$.*

Lemma 2.1 can be easily proved by explicitly expanding $\left(\mathbf{A}^l\right)_{ji}$ in terms of the entries of $\mathbf{A}$.

Definition 2.4 (Connected Nodes)**.** *In a graph $\mathsf{G}(\mathcal{V},\mathcal{E},W)$ with the adjacency matrix denoted by $\mathbf{A}$, a node $i \in \mathcal{V}$ is connected to another node $j \in \mathcal{V}$ if there exists a path starting at node i and ending at node j, i.e. $\exists l \in \mathbb{N}$ such that $\left(\mathbf{A}^l\right)_{ji} > 0$.*

Definition 2.5 (Geodesic Distance)**.** *In a graph $\mathsf{G}(\mathcal{V},\mathcal{E})$, the geodesic distance from a node $i \in \mathcal{V}$ to a connected node $j \in \mathcal{V}$ is the length of the shortest path starting at i and ending at j:*

$$d_{\mathsf{G}}(i,j) = \min\left\{l \in \mathbb{N} : \left(\mathbf{A}^l\right)_{ji} > 0\right\}$$

For disconnected nodes (i,j), the shortest path is set to infinity: $d_{\mathsf{G}}(i,j) = +\infty$. The geodesic distance is also referred to as the shortest distance.

Definition 2.6 (L-hop Neighborhood)**.** *In an undirected graph $\mathsf{G}(\mathcal{V},\mathcal{E})$, the L-hop neighborhood of a node $i \in \mathcal{V}$ is the set of nodes it is exactly L-hops away from:*

$$\mathcal{S}^{(L)}(i) = \{j \in \mathcal{V} : d_{\mathsf{G}}(j,i) = L\}$$

We will use $\mathcal{N}(i)$ to denote the 1-hop neighborhood $\mathcal{S}^{(1)}(i)$.

Definition 2.7 (Degree Matrix)**.** *The in-degree matrix of a graph is given as*

$$\mathbf{D}^{in} := diag(\mathbf{A}\mathbf{1}_N)$$

Similarly, the out-degree matrix is given as

$$\mathbf{D}^{out} := diag\left(\mathbf{A}^T\mathbf{1}_N\right)$$

For an undirected graph, $\mathbf{D}^{in} = \mathbf{D}^{out} =: \mathbf{D}$.

We extend the notation to denote the diagonal entries of the degree matrices by $d_i^{\text{in}} := \mathbf{D}_{ii}^{\text{in}}$ and $d_i^{\text{out}} := \mathbf{D}_{ii}^{\text{out}}$, $\forall i \in \mathcal{V}$. Similarly, for an undirected graph, $d_i := \mathbf{D}_{ii}$, $\forall i \in \mathcal{V}$.

Definition 2.8 (Volume of a Graph)**.** *The volume of an undirected graph $\mathsf{G}(\mathcal{V},\mathcal{E})$ is defined as the sum of the degrees of its nodes, $\mathsf{vol}(\mathsf{G}) := tr(\mathbf{D}) = 2|\mathcal{E}|$.*

Definition 2.9 (Laplacian Matrix). *The in-degree Laplacian matrix of a graph is given as*

$$\mathbf{L}^{in} := \mathbf{D}^{in} - \mathbf{A}$$

Similarly, the out-degree Laplacian matrix is given as

$$\mathbf{L}^{out} := \mathbf{D}^{out} - \mathbf{A}$$

For an undirected graph, $\mathbf{L}^{in} = \mathbf{L}^{out} =: \mathbf{L}$.

Definition 2.10 (Normalized Laplacian Matrices). *The asymmetrically normalized Laplacian matrix of an undirected graph is given as*

$$\mathbf{L}^{asym} := \mathbf{D}^{\dagger}\mathbf{L} = \mathbf{I}_N - \mathbf{D}^{\dagger}\mathbf{A}$$

where $\cdot^{\dagger}$ *denotes the Moore-Penrose pseudoinverse.* $\mathbf{L}^{asym}$ *is also called the random walk Laplacian matrix. Similarly, the symmetrically normalized Laplacian matrix is given as*

$$\mathbf{L}^{sym} := \left(\mathbf{D}^{\dagger}\right)^{1/2}\mathbf{L}\left(\mathbf{D}^{\dagger}\right)^{1/2} = \mathbf{I}_N - \left(\mathbf{D}^{\dagger}\right)^{1/2}\mathbf{A}\left(\mathbf{D}^{\dagger}\right)^{1/2}$$

Definition 2.11 (Spectral Gap). *The spectral gap of an undirected graph* $\mathsf{G}\left(\mathcal{V}, \mathcal{E}, W\right)$ *is defined as the smallest non-zero eigenvalue of* $\mathbf{L}^{sym}$. *If the graph is connected, then it is simply the second smallest eigenvalue* λ_2, *where* $0 = \lambda_1 < \lambda_2 \leq \ldots \lambda_N \leq 2$ *denote the eigenvalues of* $\mathbf{L}^{sym}$.

2.1.2 Cheeger's Inequality

The Cheeger's constant is a measure of how well-connected a graph is, particularly with respect to the bottlenecks between different parts of the graph. It captures the idea of how "easy" or "hard" it is to disconnect a graph by removing a small number of edges relative to the size of the sets being separated.

Definition 2.12 (Cheeger's Constant). *For an undirected graph* $\mathsf{G}\left(\mathcal{V}, \mathcal{E}\right)$, *the conductance* *(or the Cheeger's constant) of the graph h_{G} is defined as:*

$$h_{\mathsf{G}} = \min_{\mathcal{U} \subset \mathcal{V}} \frac{|\partial \mathcal{U}|}{\min\left\{\mathsf{vol}\left(\mathcal{U}\right), \mathsf{vol}\left(\mathcal{V} \setminus \mathcal{U}\right)\right\}}$$

where ∂ denotes the number of edges connecting the disjoint subset of $\mathcal{V}$:

$$\partial \mathcal{U} = \{(i,j) \in \mathcal{E} : (i \in \mathcal{U}, j \in \mathcal{V} \setminus \mathcal{U}) \;\; or \;\; (i \in \mathcal{V} \setminus \mathcal{U}, j \in \mathcal{U})\}$$

Note that we have extended the definition of vol *such that* $\mathsf{vol}\left(\mathcal{U}\right)$ *denotes the sum of degrees of the vertices in* $\mathcal{U}$*.*

If h_{G} is small, there is a cut that divides the graph into two parts with relatively few edges between them, i.e. the graph has a bottleneck. In the context of `GNNs`, communication between these two parts involves a large amount of information exchange via a small number of edges, which results in over-squashing; we will discuss this further in Section 2.4.

Cheeger's inequality relates the Cheeger's constant to the spectral gap of the graph:

Theorem 2.1 (Cheeger's Inequality)**.** *For an undirected graph $\mathsf{G}\left(\mathcal{V}, \mathcal{E}\right)$, the Cheeger's constant and the spectral gap are related as*

$$\frac{1}{2}\lambda_2 \leq h_{\mathsf{G}} \leq \sqrt{2\lambda_2}$$

Cheeger's inequality adds support for spatial rewiring algorithms targeted at alleviating over-squashing, which work by maximizing the spectral gap of the graphs, thereby reducing the 'bottleneckedness' in the graph.

2.1.3 Effective Resistance

Consider an unweighted and undirected graph $\mathsf{G}\left(\mathcal{V}, \mathcal{E}\right)$. If we view the graph as an electrical network with each edge representing a 1 Ohm resistor, then the *effective resistance* between nodes i and j is defined as the potential difference that would develop between them when a unit current is injected at i and drawn from j [DS84]. This is equivalent to performing

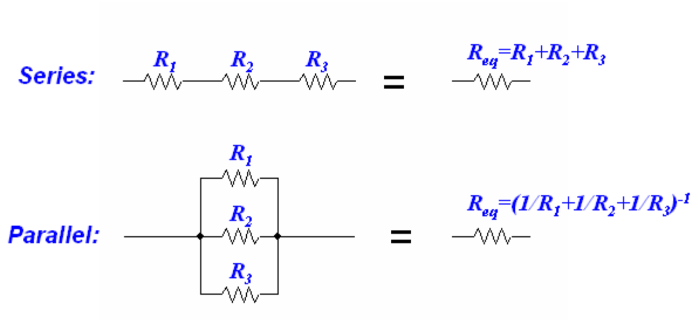


Figure 2.1: Combining resistors connected in series and parallel.

well-known series and parallel manipulations (see Figure 2.1[1]) to compute the resistance between nodes i and j in the circuit. Note that we have assumed that the nodes i and j are connected in G, i.e. there exists at least one path between them. If the nodes are disconnected, it does not make sense to talk about the effective resistance between them.

Lemma 2.2 (Effective Resistance)**.** *In an undirected graph $\mathsf{G}(\mathcal{V}, \mathcal{E})$, the effective resistance between two connected nodes $i, j \in \mathcal{V}$ is given by*

$$\mathbf{R}_{ij} = \left(\mathbf{e}^{(i)} - \mathbf{e}^{(j)}\right)^T L^\dagger \left(\mathbf{e}^{(i)} - \mathbf{e}^{(j)}\right) = L^\dagger_{ii} + L^\dagger_{jj} - 2L^\dagger_{ij}$$

where $\mathbf{e}^{(i)}$ denotes the i^{th} vector in the canonical-basis of $\mathbb{R}^N$. If the nodes are disconnected, $\mathbf{R}_{ij} = +\infty$.

The effective resistance measures how well two nodes are connected, such that *lower effective resistance suggests better connectivity.* For instance, if there are k edge-disjoint paths connecting two nodes – meaning no edge is shared between any two paths – and each path has a length at most l, then the effective resistance between those nodes will be no greater than l/k. Therefore, a high number (large k) of short paths (small l) between two nodes results in low effective resistance between them.

The following is an important result which has been used to design spacial rewiring techniques targeted at reducing over-squashing in `MPNNs` [Bla+23]:

Theorem 2.2 (Rayleigh's Monotonicity)**.** *Adding an edge to an undirected graph $\mathsf{G}(\mathcal{V}, \mathcal{E})$ can never increase the effective resistance between any two nodes.*

[1]Source: https://www.instructables.com/How-to-Change-the-Resistance-of-a-Resistor-With-An

See [Lov93, Corollary 4.3] for a proof. Clearly, the same holds for the total resistance:

Definition 2.13 (Total Resistance)**.** *The total resistance of a connected undirected graph* $\mathsf{G}(\mathcal{V}, \mathcal{E})$ *is defined as the sum of effective resistances over all node-pairs:*

$$\mathbf{R}_{tot} \coloneqq \sum_{(i,j)\in\mathcal{V}\times\mathcal{V}} \mathbf{R}_{ij}$$

2.1.4 Expected Commute Time

In order to define the expected commute time, going forward, we will consider a random walk on a connected graph $\mathsf{G}(\mathcal{V}, \mathcal{E}, W)$. The case of a disconnected graph can be dealt with by setting the commute times between disconnected nodes to ∞.

Definition 2.14 (Random Walk)**.** *A random walk on* $\mathsf{G}(\mathcal{V}, \mathcal{E}, W)$ *is a discrete time Markov process where, at each time-step, the walk transitions to an adjacent node sampled from a distribution proportional to the edge weights. In other words, it is a Markov process on the state space* $\mathcal{V}$*, with the transition matrix given by* $\mathbf{P} = \left(\mathbf{D}^{out}\right)^{\dagger}\mathbf{A}$.

Definition 2.15 (First Passage Time)**.** *The first passage time is the random variable*

$$\tau_{ij} = \min\left\{t \geq 0 | s_0 = i, s_t = j\right\}$$

The mean first passage time is given by its expectation, $\mathbf{T}_{ij} = \mathbb{E}\left[\tau_{ij}\right]$.

Definition 2.16 (First Return Time)**.** *The first return time is the random variable*

$$\tau^{+}_{ij} = \min\left\{t > 0 | s_0 = i, s_t = j\right\}$$

The mean first return time is given by its expectation, $\mathbf{T}^{+}_{ij} = \mathbb{E}\left[\tau^{+}_{ij}\right]$.

Definition 2.17 (Expected Commute Time)**.** *The expected commute time is the average number of steps it takes for a random walk to go from one node to another and then back to the source, i.e.* $\mathbf{C}_{ij} \coloneqq \mathbf{T}_{ij} + \mathbf{T}_{ji}$.

To keep the analysis simple, we will assume the graph is undirected. In this case, the pairwise expected commute times and effective resistances in a graph are intricately related [Cha+89, Theorem 2.1]:

Theorem 2.3. *The expected commute time between a pair of nodes in an undirected graph* $\mathsf{G}(\mathcal{V},\mathcal{E})$ *is proportional to the effective resistance between them:*

$$\mathbf{C}_{ij} = \mathsf{vol}(\mathsf{G})\,\mathbf{R}_{ij} = \mathsf{vol}(\mathsf{G})\left(\mathbf{L}^{\dagger}_{ii} + \mathbf{L}^{\dagger}_{jj} - 2\mathbf{L}^{\dagger}_{ij}\right)$$

2.2 Message-Passing Neural Networks

Graph Neural Networks (`GNNs`) form a special class of functions that operate on inputs of the form $(\mathsf{G},\mathbf{X})$[2], where G encodes the graph topology and $\mathbf{X} \in \mathbb{R}^{N\times H^{(0)}}$ collects the node features, each of size $H^{(0)}$. The direction of edges indicate the direction of *influence*, i.e. if $(i,j) \in \mathcal{E}$, then the representation of node j will be updated using the representation of node i in the previous layer. The output of a `GNN` depends on whether the task is node-level or graph-level. For node-level tasks, L-layer `GNNs`, parameterized by the weights θ, are of the form $\mathsf{GNN}_\theta : (\mathsf{G},\mathbf{X}) \mapsto \mathbf{Y} \in \mathbb{R}^{N\times H^{(L)}}$. For graph level tasks, they are of the form $\mathsf{GNN}_\theta : (\mathsf{G},\mathbf{X}) \mapsto \mathbf{y} \in \mathbb{R}^{H^{(L)}}$. For simplicity, we'll assume that G is unweighted and undirected.

Message-passing neural networks (`MPNNs`) [Gil+17] are a special class of `GNNs` which recursively aggregate information from the 1-hop neighborhood of each node (Definition 2.6) using *message-passing layers* of the form

$$\begin{aligned}\mathbf{z}_i^{(l)} &= \mathsf{MPNN}^{(l)}\left(\mathbf{Z}^{(l-1)},\mathsf{G}\right)\\ &= \mathsf{Upd}^{(l)}\left(\mathbf{z}_i^{(l-1)}, \mathsf{Agg}^{(l)}\left(\mathbf{z}_i^{(l-1)}, \left\{\mathbf{z}_j^{(l-1)} : j \in \mathcal{N}(i)\right\}\right)\right); \quad \mathbf{z}_i^{(0)} := \mathbf{x}_i\end{aligned} \tag{2.1}$$

where $\mathsf{Agg}^{(l)}$ denotes the *aggregation function*, $\mathsf{Upd}^{(l)}$ denotes the node *update function*, and L is the depth of the `MPNN`. Since there is no specific ordering of nodes in a graph, $\mathsf{Agg}^{(l)}$ must be *permutation invariant* in the second argument, i.e. the ordering of the neighboring nodes shouldn't affect the aggregation output. For node-level tasks, a *readout function* processes the final node representations individually to make predictions:

$$f_\theta(\mathsf{G},\mathbf{X}) = \left\{\mathsf{Out}\left(\mathbf{z}_i^{(L)}\right) : i \in \mathcal{V}\right\}$$

[2] To keep things simple, we will ignore edge features in this thesis.

For graph-level tasks, a permutation-invariant readout layer aggregates the final node representations and makes a prediction:

$$f_\theta\left(\mathsf{G}, \mathbf{X}\right) = \mathsf{Out}\left(\left\{\mathbf{z}_i^{(L)} : i \in \mathcal{V}\right\}\right)$$

Common choices for the graph-level readout include feature-wise MAX, AVG or SUM pooling layers; these are also common choices for downsampling in CNNs [GK20]. Of course, these could be composed with other differentiable mappings to get a graph-level prediction.

Since an `MPNN` recursively aggregates information from the 1-hop neighborhood of each node, an L-layer `MPNN` exposes a node to other nodes up to L-hops away.

Definition 2.18 (Receptive Field)**.** *The receptive field of an L-layer `MPNN` at node $i \in \mathcal{V}$ is given as*

$$\mathcal{B}^{(L)}\left(i\right) := \bigcup_{l=0}^{L} \mathcal{S}^{(l)}\left(i\right) = \left\{j \in \mathcal{V} : d_{\mathsf{G}}\left(j, i\right) \leq L\right\}$$

We use $\tilde{\mathcal{N}}\left(i\right)$ to denote $\mathcal{B}^{(1)}\left(i\right) = \mathcal{N}\left(i\right) \cup \{i\}$, i.e. the set of nodes whose representations are used to update the representation of node i in each layer.

This implies that nodes outside the receptive field at node i do not influence its final representation. Accordingly, an `MPNN` is said to *under-reach* if the task entails communication of information between distant nodes, which the `MPNN` simply cannot facilitate [Bar+20].

2.2.1 Graph Convolutional Networks

[KW17] introduced the Graph Convolutional Network (`GCNs`), the first successfully extension of the concept of convolution to graph-structured data. Its layer-wise propagation rule is given as

$$\mathbf{Z}^{(l)} := \begin{bmatrix} \mathbf{z}_1^{(l)} \\ \vdots \\ \mathbf{z}_N^{(l)} \end{bmatrix} = \sigma\left(\hat{\mathbf{A}}\mathbf{Z}^{(l-1)}\mathbf{W}^{(l)}\right); \quad \mathbf{Z}^{(0)} := \mathbf{X} \tag{2.2}$$

where $\hat{\mathbf{A}}$ is a *graph shift operator*, i.e. $\hat{\mathbf{A}}_{ij} \neq 0$ if and only if $(j, i) \in \tilde{\mathcal{E}} := \mathcal{E} \cup \{(k, k) : k \in \mathcal{V}\}$; it is easy to see that the `GCN` is an `MPNN`. We will refer to $\hat{\mathbf{A}}$ as the *propagation matrix*.

One popular choice for $\hat{\mathbf{A}}$ is to use a *symmetrically normalized augmented-adjacency matrix* [KW17], which ensures that the information flow remains balanced across nodes, improving gradient flow and convergence:

$$\begin{aligned}
\tilde{\mathbf{A}} &= \mathbf{A} + \mathbf{I}_N \\
\tilde{\mathbf{D}}^{\text{in}} &= \operatorname{diag}\left(\tilde{\mathbf{A}}\mathbf{1}_N\right) = \mathbf{D}^{\text{in}} + \mathbf{I}_N \\
\tilde{\mathbf{D}}^{\text{out}} &= \operatorname{diag}\left(\tilde{\mathbf{A}}^T\mathbf{1}_N\right) = \mathbf{D}^{\text{out}} + \mathbf{I}_N \\
\hat{\mathbf{A}}^{\text{sym}} &= \left(\tilde{\mathbf{D}}^{\text{in}}\right)^{-1/2} \tilde{\mathbf{A}} \left(\tilde{\mathbf{D}}^{\text{out}}\right)^{-1/2}
\end{aligned} \tag{2.3}$$

In this case, the `GCN` corresponds to an `MPNN` with the layer-wise aggregation functions and update functions given as

$$\begin{aligned}
\mathsf{Agg}^{(l)}\left(\mathbf{z}_i^{(l-1)}, \left\{\mathbf{z}_j^{(l-1)} : j \in \mathcal{N}(i)\right\}\right) &= \sum_{j \in \tilde{\mathcal{N}}(i)} \frac{1}{\sqrt{\tilde{d}_i^{\text{in}} \tilde{d}_j^{\text{out}}}} \mathbf{z}_j^{(l-1)} =: \mathbf{m}_i^{(l)} \\
\mathsf{Upd}^{(l)}\left(\mathbf{z}_i^{(l-1)}, \mathbf{m}_i^{(l)}\right) &= \sigma\left(\mathbf{W}^{(l)}\mathbf{m}_i^{(l)}\right)
\end{aligned}$$

Another choice for propagation matrix is an *asymmetrically normalized augmented-adjacency matrix* [HYL17], which corresponds to the standard form of Laplace smoothing [Tau95]:

$$\hat{\mathbf{A}}^{\text{asym}} = \left(\tilde{\mathbf{D}}^{\text{in}}\right)^{-1} \tilde{\mathbf{A}} \tag{2.4}$$

In this case, only the aggregation rule changes, and is given by

$$\mathsf{Agg}^{(l)}\left(\mathbf{z}_i^{(l-1)}, \left\{\mathbf{z}_j^{(l-1)} : j \in \mathcal{N}(i)\right\}\right) = \sum_{j \in \tilde{\mathcal{N}}(i)} \frac{1}{\tilde{d}_i^{\text{in}}} \mathbf{z}_j^{(l-1)}$$

which simply corresponds to mean aggregation of the incoming messages. Several influential works have adopted this propagation rule [HYL17; Sch+17; LHW18], and we will use it for our theoretical analysis in Section 3.2.

2.2.2 Graph Attention Networks

One shortcoming in the `GCN` architecture is that it uses only the degree distribution to compute the weights for message aggregation; this limits its representational power. The Graph Attention Network (GAT) [Vel+18] overcomes this by using attention mechanism for computing weights from node features, allowing for more context-sensitive feature aggregation. Formally, the aggregation functions of `GAT` are given as

$$\mathsf{Agg}^{(l)}\left(\mathbf{z}_i^{(l-1)}, \left\{\mathbf{z}_j^{(l-1)} : j \in \mathcal{N}(i)\right\}\right) = \sum_{j \in \tilde{\mathcal{N}}(i)} \alpha_{ij}^{(l)} \mathbf{z}_j^{(l-1)} \tag{2.5}$$

where α_{ij} are the learnt attention scores, given by

$$\begin{aligned} \alpha_{ij}^{(l)} &= \left[\text{softmax}\left(\left\{\mathsf{LeakyReLU}\left\langle \mathbf{a}, \left[\mathbf{W}^{(l)}\mathbf{z}_i^{(l-1)} \middle\| \mathbf{W}^{(l)}\mathbf{z}_k^{(l-1)}\right]\right\rangle : k \in \tilde{\mathcal{N}}(i)\right\}\right)\right]_j \\ &= \frac{\mathsf{LeakyReLU}\left\langle \mathbf{a}, \left[\mathbf{W}^{(l)}\mathbf{z}_i^{(l-1)} \middle\| \mathbf{W}^{(l)}\mathbf{z}_j^{(l-1)}\right]\right\rangle}{\sum_{k\in\tilde{\mathcal{N}}(i)} \mathsf{LeakyReLU}\left\langle \mathbf{a}, \left[\mathbf{W}^{(l)}\mathbf{z}_i^{(l-1)} \middle\| \mathbf{W}^{(l)}\mathbf{z}_k^{(l-1)}\right]\right\rangle} \end{aligned}$$

Here, $\|$ represents a concatenation, $\mathbf{a} \in \mathbb{R}^{2H^{(l)}}$ is a learnable vector, and the `LeakyReLU` activation [MHN13] is

$$\mathsf{LeakyReLU}(x;\gamma) = \begin{cases} x, & x > 0 \\ \gamma x, & x \le 0 \end{cases} \tag{2.6}$$

for some fixed $\gamma \in (0,1)$, called the *slope* (hyper-)parameter. Like in [Vas+17], the self-attention mechanism can be stabilized using multiple attention heads:

$$\begin{aligned} \mathsf{Agg}^{(l,k)}\left(\left\{\mathbf{z}_j^{(l-1)} : j \in \tilde{\mathcal{N}}(i)\right\}\right) &= \sum_{j\in\tilde{\mathcal{N}}(i)} \alpha_{ij}^{(l,k)} \mathbf{z}_j^{(l-1)} =: \mathbf{m}_i^{(l,k)}, \quad k \in [K] \\ \mathsf{Upd}^{(l)}\left(\mathbf{z}_i^{(l-1)}, \left\{\mathbf{m}_i^{(l,k)}\right\}_{k=1}^{K}\right) &= \mathop{\Big\|}_{k=1}^{K} \sigma\left(\mathbf{W}^{(l,k)}\mathbf{m}_i^{(l,k)}\right) \end{aligned}$$

In our experiments, we set $K = 2$ in order to keep the computational load manageable, while at the same time harnessing the expressiveness of the multi-headed self-attention mechanism.

2.3 DropEdge

DropEdge [Ron+20] was originally introduced to tackle the problems of *over-fitting* and *over-smoothing* [LHW18; OS20], which limit the trainability of a deep GNN. It tackles both of these problems together, allowing practitioners to train deeper GNNs on important graph learning tasks.

2.3.1 The Algorithm

DropEdge is a random data augmentation technique that works by sampling a subgraph of the original input graph, and using that for message passing [Ron+20]. Specifically, in each forward pass, DropEdge perturbs the adjacency matrix by randomly dropping some edges, independently and identically from a Bernoulli distribution with probability q. Accordingly, the augmented adjacency matrix is given by

$$\begin{aligned} \mathbf{M} &\sim \{\text{Bern}\,(1-q)\}^{N\times N} \\ \tilde{\mathbf{A}} &= \mathbf{M} \circ \mathbf{A} + \mathbf{I}_N \end{aligned} \tag{2.7}$$

where $\circ$ denotes the Hadamard product (element-wise multiplication). Then, the forward propagation is performed with this perturbed adjacency matrix, as usual. DropEdge reduces over-fitting by randomly selecting a subset of neighbors for message aggregation, which introduces uncertainty that prevents the model from relying heavily on the features of specific neighbors. It also helps alleviate over-smoothing by reducing the number of messages being propagated in the graph. At test-time, the standard practice is to simply turn DropEdge off; if a normalized adjaceny matrix is used for propagation, then no re-normalization of weights or activations is needed [Ron+20] (unlike in the case of common dropping methods [Sri+14]).

We refer to the DropEdge variant in Equation 2.7 as *one-shot* DropEdge – it corresponds to using the same perturbed adjacency matrix in all layers. Indeed, the training can be made noisier by sampling a different adjacency matrix for each layer:

$$\begin{aligned}\mathbf{M}^{(l)} &\sim \{\text{Bern}\,(1-q)\}^{N\times N}\\ \tilde{\mathbf{A}}^{(l)} &= \mathbf{M}^{(l)} \circ \mathbf{A} + \mathbf{I}_N\end{aligned} \tag{2.8}$$

In practice, *layer-wise* `DropEdge` outperforms one-shot `DropEdge` on the training set, but the performance is comparable on the test set [Ron+20, Section 5.2.3].

The common practice of turning `DropEdge` off during test-time has been shown to raise the over-smoothing levels back up [XZL23, Figure 1]. Inspired by the Monte-Carlo Dropout [GG16], another way of making predictions on the test-set is to perform multiple stochastic forward passes with `DropEdge` turned on, and then averaging the predictions – we call this the *Monte Carlo DropEdge* (MC-DE). On top of alleviating over-smoothing, this approach also performs better than the regular test-time implementation [XZL23, Table 1].

2.3.2 Effect on Over-smoothing

Over-smoothing is a problem specific to `GNNs`, and it hinders model training by causing the output representations to become independent of the input features, as the network depth increases [LHW18; OS20]. The over-smoothing problem was first highlighted in [LHW18], where they showed that in the infinite width limit of a linear `GCN`, $L \to \infty$, the node representations in a connected component, $\mathcal{U} \subset \mathcal{V}$, converge to a fixed point in $\mathbb{R}^{H^{(\infty)}}$, which is determined only by the corresponding degree set $\left\{d_1, \ldots, d_{|\mathcal{U}|}\right\}$. This is clearly detrimental to the performance of `GCNs`, since the information in the node features is completely lost. [OS20] extended these results to `ReLU`[3] networks with convolution filters. They characterize over-smoothing as the exponentially fast convergence of the node representations to a special kind of subspace, $\mathcal{M} \subset \mathbb{R}^{N\times H^{(\infty)}}$, instead of a fixed point.

Definition 2.19 (Subspace)**.** *Let $\mathcal{M} := \left\{\mathbf{UC} \middle| \mathbf{C} \in \mathbb{R}^{M\times H}\right\} \subset \mathbb{R}^{N\times H}$ be an M-dimensional subspace of $\mathbb{R}^{N\times H}$, where $\mathbf{U} \in \mathbb{R}^{N\times M}$ is orthogonal, i.e. $\mathbf{U}^T\mathbf{U} = \mathbf{I}_M$, and $M \leq N$.*

Definition 2.20 (ϵ-smoothing)**.** *We say that the node representations of a `GCN` are ϵ-smoothed if there exists an $L \in \mathbb{N}$ such that $\forall l \geq L$, the hidden representation matrices,*

[3] The `ReLU` activation is given by $\texttt{ReLU} = \texttt{LeakyReLU}\,(\cdot; 0)$, and `LeakyReLU` is as in Equation 2.6.

$\mathbf{Z}^{(l)}$, *are at most* $\epsilon > 0$ *distance away from a subspace* $\mathcal{M}$ *that does not depend on the input features* $\mathbf{X}$*:*

$$d_{\mathcal{M}}\left(\mathbf{Z}^{(l)}\right) := \inf_{\mathbf{M}\in\mathcal{M}} \left\|\mathbf{Z}^{(l)} - \mathbf{M}\right\|_F < \epsilon, \quad \forall l \geq L \tag{2.9}$$

where $\|\cdot\|_F$ *denotes the Frobenius norm.*

Definition 2.21 (Relaxed ϵ-smoothing Layer)**.** *Assume that the weights of a* `ReLU-GCN` *are initialized such that their spectral norms are bounded, i.e.* $s_l := \left\|\mathbf{W}^{(l)}\right\|_2 \leq 1$, $\forall l$*. Given the subspace* $\mathcal{M}$ *and* ϵ*, we define the relaxed smoothing layer as*

$$\hat{L}\left(\mathcal{M}, \epsilon\right) := \left\lceil \frac{\log\left(\epsilon / d_{\mathcal{M}}\left(\mathbf{X}\right)\right)}{\log s\lambda} \right\rceil$$

where $d_{\mathcal{M}}$ *is as in Equation 2.9,* $s = \sup_{l\in\mathbb{N}} s_l$*, and* $\lambda < 1$ *is the second largest eigenvalue of the propagation matrix* $\hat{\mathbf{A}} = \hat{\mathbf{A}}^{sym}$ *(Equation 2.3).*

`DropEdge` can be viewed as a *message passing reducer*, i.e. it makes the graph topology sparser, thereby reducing the extent of over-smoothing in deep `GCNs`. Following the setup in [OS20], [Ron+20, Theorem 1] showed that using `DropEdge` allows for a higher number of message-passing steps before a preset level of over-smoothing is reached:

Theorem 2.4 (DropEdge Reduces Over-smoothing)**.** *Consider the input graph* $\mathsf{G}\left(\mathcal{V}, \mathcal{E}\right)$ *and the subgraph* $\mathsf{G}'\left(\mathcal{V}, \mathcal{E}'\right)$ *obtained by removing some edges. Furthermore, let* $\mathcal{M}$ *and* $\mathcal{M}'$ *denote the subspaces these graphs will encounter* ϵ*-smoothing with respect to. Then, after a sufficient number of edges have been removed, one of the following holds:*

- *The relaxed smoothing layer increases,* $\hat{L}\left(\mathcal{M}, \epsilon\right) \leq \hat{L}\left(\mathcal{M}', \epsilon\right)$
- *The information loss decreases,* $N - dim\left(\mathcal{M}\right) > N - dim\left(\mathcal{M}'\right)$

Although the exact details of the proof of Theorem 2.4 are not relevant to our work, we wish to highlight that it relies on the fact that removing edges from a graph can only increase the effective resistance between any two nodes (Theorem 2.2). In a way, `DropEdge` reduces over-smoothing by increasing the effective resistances in the graph. In Section 2.4, we will see how this observation hints at the negative effects of `DroEdge` on over-squashing.

[Ron+20] showed that `DropEdge` improves the performance of deep `GNNs` on a range of graph learning tasks, including the citation networks `Cora` [McC+00], `CiteSeer` [GBL98] and `PubMed` [Nam+12], as well as the `Reddit` social network [HYL17]. Moreover, it has inspired the design of several newer dropping methods, like DropNode [Fen+20], DropGNN [Pap+21], DropAGG [Jia+23], DropMessage [Fan+23] and Structure-Aware DropEdge [Han+23], which have significantly improved the performance of `GNNs`. Given its widespread impact on `GNN` research, it is only natural to question its efficacy in different settings, including its performance on long-range tasks.

2.4 Over-squashing

Another problem inherent to the task of modelling graph data using `MPNNs` is *over-squashing*. In simple words, the topology of the input graph can result in *bottlenecks*, which causes information from exponentially growing neighborhoods [CZS18] to be *squashed* into finite-sized node representations [AY21]. This results in a loss of information as it is propagated over long distances. Therefore, the `MPNNs` fail to capture LRIs in the graph, limiting their applications to short-range tasks. On a variety of (synthetic and real-world) long-range datasets, [AY21] showed that the `MPNNs` suffer from underfitting when the *problem radius*, i.e. the range of interaction in the ground-truth, is large and the depth of the network is comparable to it.

[Top+22; Bla+23; Di +23; Gio+24] have shown that over-squashing in a `GNN` is intricately related to topological properties of the graph, such as the *curvature* of its edges, the *effective resistance* between pairs of nodes and the *expected commute time* between them. Optimizing for such properties, several graph rewiring techniques have successfully reduced over-squashing [Arn+22; DLV22; Top+22; Bla+23; Gir+23; KBM23], improving the performance of `GNNs` on long-range tasks. Accordingly, in this section, we will introduce the theoretical results characterizing the relationships between over-squashing and the input graph's topology. This will help motivate our theoretical analysis in Chapter 3.

2.4.1 Sensitivity

[AY21] empirically showed that `MPNNs` fail to communicate information effectively over long distances, which results in poor performance on long-range tasks – a phenomenon they coined 'over-squashing'. These findings were theoretically studied in [Top+22], wherein over-squashing was characterized in terms of the Jacobian of the `MPNN`'s node-level representations w.r.t. the input features:

Definition 2.22 (1-norm)**.** *The 1-norm of a matrix* $\mathbf{Q} \in \mathbb{R}^{D_1 \times D_2}$ *is defined as*

$$\left\|\mathbf{Q}\right\|_1 = \sum_{i=1}^{D_1} \sum_{j=1}^{D_2} \left|\mathbf{Q}_{ij}\right|$$

Definition 2.23 (Sensitivity)**.** *Consider an L-layer `MPNN` and its input* $\left(\mathsf{G}\left(\mathcal{V}, \mathcal{E}\right), \mathbf{X}\right)$. *The sensitivity of a node* $i \in \mathcal{V}$ *to another node* $j \in \mathcal{V}$ *is defined as* $\left\|\partial \mathbf{z}_i^{(L)} / \partial \mathbf{x}_j\right\|_1$.

Over-squashing in an `MPNN` can be understood as low sensitivity between distant nodes, i.e. small perturbations in a node's features don't effect other distant nodes' representations. [Top+22] showed that the sensitivity between two nodes can be characterized using the propagation matrix:

Lemma 2.3. *Consider an L-layer `MPNN` with the aggregation functions given by*

$$\mathsf{Agg}^{(l)}\left(\mathbf{z}_i^{(l-1)}, \left\{\mathbf{z}_j^{(l-1)} : j \in \mathcal{N}\left(i\right)\right\}\right) = \sum_{j \in \tilde{\mathcal{N}}(i)} \hat{\mathbf{A}}_{ij} \mathsf{Msg}^{(l)}\left(\mathbf{z}_i^{(l-1)}, \mathbf{z}_j^{(l-1)}\right)$$

where $\mathsf{Msg}^{(l)}$ *denotes a differentiable message function. If the gradients of the message and the update functions are bounded as* $\left\|\nabla \mathsf{Msg}^{(l)}\right\|_1 \leq \alpha$ *and* $\left\|\nabla \mathsf{Upd}^{(l)}\right\|_1 \leq \beta$, $\forall l \in [L]$, *then the sensitivity of a node* $i \in \mathcal{V}$ *to another node* $j \in \mathcal{S}^{(L)}\left(i\right)$ *satisfies*

$$\left\|\frac{\partial \mathbf{z}_i^{(L)}}{\partial \mathbf{x}_j}\right\|_1 \leq \left(\alpha\beta\right)^L \left(\hat{\mathbf{A}}^L\right)_{ij}$$

In other words, given an `MPNN` with bounded gradients, the sensitivity between two nodes can be studied using the corresponding entries of the matrix $\hat{\mathbf{A}}^L$. [Bla+23, Lemma

3.2] extended these results to all pairs of nodes in the graph, but assuming a *strictly less general* form of the aggregation function:

Lemma 2.4. *Consider an L-layer* `MPNN` *with the aggregation functions given by*

$$\mathsf{Agg}^{(l)}\left(\mathbf{z}_i^{(l-1)}, \left\{\mathbf{z}_j^{(l-1)} : j \in \mathcal{N}(i)\right\}\right) = \sum_{j \in \tilde{\mathcal{N}}(i)} \hat{\mathbf{A}}_{ij} \mathsf{Msg}^{(l)}\left(\mathbf{z}_j^{(l-1)}\right)$$

where $\mathsf{Msg}^{(l)}$ *denotes a differentiable message function*[4]*. If the gradients of the message and the update functions are bounded*[5] *as* $\left\|\nabla \mathsf{Msg}^{(l)}\right\|_1 \leq \alpha$ *and* $\max\left\{\left\|\nabla \mathsf{Upd}^{(l)}\right\|_1, 1\right\} \leq \beta$, $\forall l \in [L]$*, then the sensitivity of a node* $i \in \mathcal{V}$ *to another node* $j \in \mathcal{V}$ *satisfies*

$$\left\|\frac{\partial \mathbf{z}_i^{(L)}}{\partial \mathbf{x}_j}\right\|_1 \leq (2\alpha\beta)^L \sum_{l=0}^{L} \left(\hat{\mathbf{A}}^l\right)_{ij}$$

Let's do a sanity check: since $\hat{\mathbf{A}}_{ij} \neq 0$ if and only if $(i,j) \in \tilde{\mathcal{E}}$, $\left(\hat{\mathbf{A}}^l\right)_{ij} \neq 0$ if and only if $j \in \mathcal{B}^{(l)}(i)$. Therefore, for $j \notin \mathcal{B}^{(L)}(i)$, Lemma 2.4 suggests the sensitivity of node i's representations to node j's features is 0, as expected.

Manipulating the upper bound in Lemma 2.4, [Bla+23, Theorem 3.3] bounded the sensitivity between two nodes with the effective resistance between them:

Theorem 2.5 (Sensitivity and Effective Resistance)**.** *Under the assumptions of Lemma 2.4, and with* $\hat{\mathbf{A}} = \hat{\mathbf{A}}^{sym}$,

$$\left\|\frac{\partial \mathbf{z}_i^{(L)}}{\partial \mathbf{x}_j}\right\|_1 \leq (2\alpha\beta)^L \frac{\max\{d_i, d_j\}}{2} \left(\frac{2}{\min\{d_i, d_j\}}\left(L + 1 + \frac{\mu^{L+1}}{1-\mu}\right) - \mathbf{R}_{ij}\right)$$

where $\mathbf{R}_{ij}$ *denotes the effective resistance between the nodes* $i, j \in \mathcal{V}$*, and* $\max\{|\mu_2|, |\mu_N|\} \leq \mu$*, where* $\mu_N \leq \ldots \leq \mu_2 < \mu_1 = 1$ *denote the eigenvalues of* $\hat{\mathbf{A}}^{sym}$.

One interpretation of this theorem is that when two nodes are connected by multiple short paths (which leads to low effective resistance), the input features of one node have a strong influence on the representation of the other node. This makes intuitive sense since a higher

[4]While a `GCN` is included in the class of `MPNNs` assumed in Lemma 2.3 and Lemma 2.4, a `GAT` is not.

[5][Bla+23] used the operator norm to define boundedness, but the result continues to hold for the 1-norm.

number of short paths implies that there are multiple ways of information from one node reaching the other, without getting lost due to over-squashing. The relationship between effective resistance and sensitivity in Theorem 2.5 also suggests the use of total resistance as a heuristic for the *message passing rate* in an input graph.

2.4.2 Influence Distribution

In this subsection, we will present a theoretical result which was not introduced in the context of over-squashing but has nevertheless been influential in its analysis [Xu+18]. We start with some definitions to set up the theorem:

Definition 2.24 (Augmented Random Walk)**.** *An augmented random walk on an unweighted graph* $\mathsf{G}(\mathcal{V}, \mathcal{E})$ *is a random walk on the augmented graph* $\mathsf{G}\left(\mathcal{V}, \tilde{\mathcal{E}}\right)$*, where*

$$\tilde{\mathcal{E}} \coloneqq \mathcal{E} \cup \{(k,k) : k \in \mathcal{V}\}$$

In other words, it is a Markov process on the state space $\mathcal{V}$ *with the transition matrix given by* $\tilde{\mathbf{P}} = \left(\tilde{\mathbf{D}}^{out}\right)^{-1} \tilde{\mathbf{A}}$.

Definition 2.25 (Influence Distribution)**.** *For an L-layer* `MPNN`*, the influence distribution of node i is defined as*

$$I_i(j) = \frac{I(i,j)}{\sum_{k \in \mathcal{V}} I(i,k)} = \frac{\left\| \partial \mathbf{z}_i^{(L)} / \partial \mathbf{x}_j \right\|_1}{\sum_{k \in \mathcal{V}} \left\| \partial \mathbf{z}_i^{(L)} / \partial \mathbf{x}_k \right\|_1}$$

We will assume that the input graphs are undirected and the `MPNN` is an L-layer `ReLU-GCN` with the asymmetric propagation rule (Equation 2.4):

$$\mathbf{Z}^{(l)} = \mathsf{ReLU}\left(\hat{\mathbf{A}}^{\text{asym}} \mathbf{Z}^{(l-1)} \mathbf{W}^{(l)}\right) \tag{2.10}$$

Observe that $\tilde{\mathbf{P}} = \hat{\mathbf{A}}^{\text{asym}}$, since the in-degree and out-degree matrices of an undirected graph are equal (Definition 2.7).

Recall the definition of the ReLU non-linearity:

$$\mathsf{ReLU}(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

Clearly, the ReLU activation blocks the flow of gradients when the pre-activation value x is non-positive[6]. Accordingly, we say that a neuron is *active* if the pre-activation it stores is positive, else it is *inactive*. Similarly, any path in the (directed acyclic) computation graph of a GNN is *blocked* if some neuron on the path is inactive.

Assumption 2.1. *Say there are Ψ_{ij} paths in the computational graph going from the input features $\mathbf{x}_j$ to the final node representations $\mathbf{z}_i^{(L)}$. Assume that each of these paths is active with the same probability ρ, which is independent of the input features $\mathbf{X}$ and the model parameters $\mathbf{W}^{(1)}, \dots, \mathbf{W}^{(L)}$.*

This assumption was introduced to analyze the local minima of ReLU-MLPs [Kaw16], and has been recently used to study over-squashing in MPNNs [Di +23]. Moreover, under this assumption, [Xu+18] showed that the *influence distribution* of a node i is equal to the L-step transition probabilities in the augmented random walk.

Theorem 2.6. *Given the MPNN in Equation 2.10, and under Assumption 2.1,*

$$\mathbb{E}\left[\frac{\partial \mathbf{z}_i^{(L)}}{\partial \mathbf{x}_j}\right] = \left(\left(\hat{\mathbf{A}}^{asym}\right)^L\right)_{ij}\left[\rho \prod_{l=1}^{L} \mathbf{W}^{(l)}\right] = \left(\tilde{\mathbf{P}}^L\right)_{ij}\left[\rho \prod_{l=1}^{L} \mathbf{W}^{(l)}\right]$$

Corollary 2.1. *Since $\mathbb{E}\left[\partial \mathbf{z}_i^{(L)} / \partial \mathbf{x}_j\right]$ is proportional to $\left(\tilde{\mathbf{P}}^L\right)_{ij}$, the influence distribution is given by $I_i(j) = \left(\tilde{\mathbf{P}}^L\right)_{ij}$.*

2.4.3 Jacobian Obstruction

While the bounds in the previous subsections provide important insights into the relationship between sensitivity and the graph topology, they don't allow us to reliably predict how much more sensitive a node's representation is to its own features than to its neighbors'

[6]Although ReLU is not differentiable at $x = 0$, a sub-gradient $v \in \partial\mathsf{ReLU}(0) = [0, 1]$ can be used instead for optimization; the PyTorch implementation uses $v = 0$.

features. To that end, we will present the *Jacobian obstruction* as a measure of excess sensitivity.

In this section, we will assume the GNN to be an L-layer MPNN with the aggregation functions of the form

$$\mathsf{Agg}^{(l)}\left(\mathbf{z}_i^{(l-1)}, \left\{\mathbf{z}_j^{(l-1)} : j \in \mathcal{N}(i)\right\}\right) = \sum_{j \in \tilde{\mathcal{N}}(i)} \hat{\mathbf{A}}_{ij}\mathbf{z}_j^{(l-1)}$$

and the update functions of the form

$$\begin{aligned}
\mathbf{z}_i^{(l)} &= \mathsf{Upd}^{(l)}\left(\mathbf{z}_i^{(l-1)}, \mathsf{Agg}^{(l)}\left(\mathbf{z}_i^{(l-1)}, \left\{\mathbf{z}_j^{(l-1)} : j \in \mathcal{N}(i)\right\}\right)\right) \\
&= \mathsf{ReLU}\left(\mathbf{W}^{(l)}\left(c_r\mathbf{z}_i^{(l-1)} + c_a\mathsf{Agg}^{(l)}\left(\mathbf{z}_i^{(l-1)}, \left\{\mathbf{z}_j^{(l-1)} : j \in \mathcal{N}(i)\right\}\right)\right)\right) \\
&= \mathsf{ReLU}\left(\mathbf{W}^{(l)}\left(c_r\mathbf{z}_i^{(l-1)} + c_a\sum_{j \in \tilde{\mathcal{N}}(i)} \hat{\mathbf{A}}_{ij}\mathbf{z}_j^{(l-1)}\right)\right) \qquad (2.11)
\end{aligned}$$

where $c_r \geq c_a > 0$, $\hat{\mathbf{A}} = \hat{\mathbf{A}}^{\text{sym}}$ (Equation 2.3) and $\mathbf{W}^{(l)}$ are weight matrices of appropriate size.

Consider the following quantity which measures the excess sensitivity of the final representation of node i to some intermediate representation of node i compared to the corresponding intermediate representation of node j [Di +23]:

$$\mathbf{J}^{(l)}(i,j) := \frac{1}{d_i}\frac{\partial \mathbf{z}_i^{(L)}}{\partial \mathbf{z}_i^{(l)}} - \frac{1}{\sqrt{d_i d_j}}\frac{\partial \mathbf{z}_i^{(L)}}{\partial \mathbf{z}_j^{(l)}}; \quad l \in [L]$$

The normalization stems from our choice of the propagation matrix. The *reverse triangle inequality* tells us that

$$\begin{aligned}
\left\|\mathbf{J}^{(l)}(i,j)\right\| &\geq \left|\left\|\frac{1}{d_i}\frac{\partial \mathbf{z}_i^{(L)}}{\partial \mathbf{z}_i^{(l)}}\right\|_1 - \left\|\frac{1}{\sqrt{d_i d_j}}\frac{\partial \mathbf{z}_i^{(L)}}{\partial \mathbf{z}_j^{(l)}}\right\|_1\right| \\
&\geq \frac{1}{d_i}\left\|\frac{\partial \mathbf{z}_i^{(L)}}{\partial \mathbf{z}_i^{(l)}}\right\|_1 - \frac{1}{\sqrt{d_i d_j}}\left\|\frac{\partial \mathbf{z}_i^{(L)}}{\partial \mathbf{z}_j^{(l)}}\right\|_1
\end{aligned}$$

If the `MPNN` cannot effectively propagate information from node j to node i, we would expect $\mathbf{z}_i^{(L)}$ to be more sensitive to $\mathbf{z}_i^{(l)}$ than to $\mathbf{z}_j^{(l)}$ [Di +23, Theorem 4.1], i.e. $\left\|\partial\mathbf{z}_i^{(L)}/\partial\mathbf{z}_i^{(l)}\right\|_1 \gg \left\|\partial\mathbf{z}_i^{(L)}/\partial\mathbf{z}_j^{(l)}\right\|_1$. In that case, without making any assumptions about d_i and d_j, we would expect the second inequality above to be an equality, so that the norm of $\mathbf{J}^{(l)}(i,j)$ is an upper bound on the excess sensitivity. This makes $\mathbf{J}^{(l)}(i,j)$ suitable for measuring over-squashing. Indeed, we can follow a similar intuition to define a symmetric quantity measuring the 2-way communication between a pair of nodes [Di +23]:

$$\tilde{\mathbf{J}}^{(l)}(i,j) \coloneqq \mathbf{J}^{(l)}(i,j) + \mathbf{J}^{(l)}(j,i)$$

Definition 2.26 (Symmetric Jacobian Obstruction)**.** *In an L-layer `MPNN` of the form Equation 2.11, the symmetric Jacobian obstruction between two nodes $i, j \in \mathcal{V}$ is given by*

$$\tilde{\mathsf{O}}(i,j) = \sum_{l=0}^{L} \left\|\tilde{\mathbf{J}}^{(l)}(i,j)\right\|_1$$

Observe that $\tilde{\mathsf{O}}$ accounts for the sensitivity of the final representations w.r.t. *all* intermediate representations, and not just the input features. [Di +23, Theorem 5.5] showed that $\left\|\tilde{\mathsf{O}}(i,j)\right\|_1$ can be bounded using the effective resistance between the nodes:

Theorem 2.7 (Jacobian Obstruction and Effective Resistance)**.** *Assume that the layer-wise updates of the `GNN` are as in Equation 2.11, and Assumption 2.1 holds. Let*

$$\mu = \max_{l\in[L]} \left\|\mathbf{W}^{(l)}\right\|_1 \quad \textit{and} \quad \nu = \min_{l\in[L]} \left\|\mathbf{W}^{(l)}\right\|_1$$

If $\mu(c_r + c_a) \leq 1$, then $\exists \epsilon_{\mathsf{G}}$, independent of the choice of nodes, such that

$$\epsilon_{\mathsf{G}}(1 - o(L)) \left(\frac{\rho}{\nu c_a}\right) \mathbf{R}_{ij} \leq \tilde{\mathsf{O}}(i,j) \leq \left(\frac{\rho}{\mu c_a}\right) \mathbf{R}_{ij}$$

where $o(L)$ denotes a term decaying exponentially in L.

This result can be interpreted as follows: higher effective resistance between two nodes implies a large Jacobian obstruction, i.e. the nodes have low (relative) sensitivity to each other. Importantly, the bounds become tighter as L grows. Therefore, in deep `MPNNs`, the effective resistance can reliably characterize the over-squashing levels.

2.5 Related Works

This section is dedicated to the discussion of works that are adjacent to ours. These are not too important for understanding our contributions, but are relevant for understanding their significance. In Section 2.5.1, we present algorithms designed for alleviating over-smoothing and training deeper `GNNs`. The purpose of this subsection is to present a list of methods that can benefit from a similar analysis as we perform for `DropEdge` in this work. In Section 2.5.2, we present rewiring techniques developed for addressing over-squashing. The purpose is to highlight a common principle in all these methods, which is that edge addition is necessary, or rather, edge deletion by itself is likely harmful – this points towards a need for evaluating `DropEdge` on long-range tasks. In Section 2.5.3, we present methods designed for a unified treatment of the two problems. This is to highlight the trade-off between over-smoothing and over-squashing, and to further emphasize the need for re-evaluating the methods in Section 2.5.1. Furthermore, the techniques in this section can inspire future works aimed at adapting the algorithms in Section 2.5.1 to simultaneously address the over-squashing problem. Finally, in Section 2.5.4, we present a collection of long range graph benchmarks that can be used to reliably conclude the efficacy of methods designed for tackling the over-smoothing and over-squashing problems.

2.5.1 Treating Over-smoothing

Over-smoothing prevents deep `MPNNs` from learning informative node representations that can predict the true labels with a good precision. This can limit how deep architectures can go, keeping them from modelling LRIs in, e.g. *heterophilic* graphs, wherein the labels of neighboring nodes tend to differ [Zhu+20]. In an effort to overcome such limitations, a large set of tools, including `DropEdge` [Ron+20], have been proposed for alleviating over-smoothing; we will introduce some of them in this section. An analysis of their effects on over-squashing is warranted, similar to the one we will perform for `DropEdge`.

We will start by quickly introducing two efficiently computable measures of over-smoothing. The first is based on the concept of the *Dirichlet energy* on graphs. For representations in layer l, it is defined as

$$\mathrm{E}\left(\mathbf{Z}^{(l)}\right) = \frac{1}{N} \sum_{(i,j) \in \mathcal{E}} \left\| \mathbf{z}_i^{(l)} - \mathbf{z}_j^{(l)} \right\|_2^2$$

That is, it computes the sum of squared Euclidean distances between the representations of all adjacent node-pairs (followed by normalization). Clearly, a small Dirichlet energy would imply that the representations are too similar to each other and are therefore, likely uninformative [CW20b]. The other measure is the *Mean Average Distance* (MAD) [Che+20a], which uses the cosine distance in place of the squared Euclidean distance:

$$\mu\left(\mathbf{Z}^{(l)}\right) = \frac{1}{N} \sum_{(i,j) \in \mathcal{E}} \left(1 - \frac{\left\langle \mathbf{z}_i^{(l)}, \mathbf{z}_j^{(l)} \right\rangle}{\left\| \mathbf{z}_i^{(l)} \right\|_2 \left\| \mathbf{z}_j^{(l)} \right\|_2} \right)$$

In the case of scalar features, MAD computes to 0 as long as all the features are of the same sign. This makes MAD a problematic measure [RBM23]. Nevertheless, it is commonly used to provide empirical support for strategies developed for tackling over-smoothing.

A popular choice for reducing over-smoothing is to regularize the model. Recall that `DropEdge` implicitly regularizes the model by adding noise to the learning trajectory (Section 2.3). Graph Drop Connect (GDC) [Has+20] combines `DropEdge` and DropMessage [Fan+23] together, resulting in a layer-wise sampling scheme that uses a different subgraph for message-aggregation over each feature dimension. Another powerful form of implicit regularization is feature normalization, which has proven crucial in enhancing the performance and stability of several types of neural networks [Hua+20]. Exploiting the inductive bias in graph-structured data, normalization techniques like PairNorm [ZA20], Differentiable Group Normalization (DGN) [Zho+20] and NodeNorm [Zho+21b] have been proposed to reduce over-smoothing in `GNNs`. On the other hand, Energetic Graph Neural Networks (E`GNNs`) [Zho+21a] explicitly regularize the optimization by constraining the layer-wise Dirichlet energy to a predefined range.

In a different vein, motivated by the success of residual networks (ResNets) [He+16] in computer vision, [Li+19] proposed the use of residual connections to prevent the smoothing of representations:

$$\mathbf{Z}^{(l)} = \mathbf{Z}^{(l-1)} + \mathsf{MPNN}^{(l)}\left(\mathbf{Z}^{(l-1)}, \mathsf{G}\right)$$

This update rule successfully improved the performance of `GCN` [KW17] on a range of graph-learning tasks. [Che+20b] introduced `GCNII`, which uses skip connections from the input to all hidden layers. Its propagation rule is given by

$$\mathbf{Z}^{(l)} = \sigma\left[\left(\left(1-\alpha^{(l)}\right)\hat{\mathbf{A}}^{\mathrm{sym}}\mathbf{Z}^{(l-1)} + \alpha^{(l)}\mathbf{X}\right)\left(\left(1-\beta^{(l)}\right)\mathbf{I}_N + \beta^{(l)}\mathbf{W}^{(l)}\right)\right]$$

where $\alpha^{(l)}$ and $\beta^{(l)}$ are hyper-parameters. This layer wise propagation rule has allowed for training of ultra-deep networks – up to 64 layers. Impressively, `GCNII` actually benefits from increasing network depth, even on (homophilic) citation networks [YCS16]. Some other architectures, like the Jumping Knowledge Network (JKNet) [Xu+18] and the Deep Adaptive GNN (DAGNN) [LGJ20], aggregate the representations from *all* layers, $\left\{\mathbf{z}_i^{(l)}\right\}_{l=1}^{L}$, before processing them through a readout layer, Out.

2.5.2 Treating Over-squashing

Over-squashing of information being propagated over long distances impedes the capacity of an `MPNN` to model LRIs. This can be detrimental to the performance of `MPNNs` on tasks of practical importance, such as those in biochemistry [Irw+12; Ram+14; HYL17; Mor+20]. In the previous sections, we saw that over-squashing is significantly affected by the graph topology, such that it is harder for poorly connected nodes to communicate with each other. In this section, we will review some of the graph rewiring methods proposed to address the problem of over-squashing. Particularly, we wish to emphasize one commonality between all these methods – edge addition (sometimes along with edge removal) is necessary.

Graph rewiring refers to modifying the edge set of a graph by adding and removing edges in a systematic manner:

$$\mathsf{Rew}\left(\mathsf{G}\left(\mathcal{V},\mathcal{E}\right)\right) = \mathsf{G}\left(\mathcal{V}, \mathcal{E}^{\mathsf{Rew}} := \left(\mathcal{E}\setminus\mathcal{E}^{\mathsf{rem}}\right)\cup\mathcal{E}^{\mathsf{add}}\right)$$

where $\mathcal{E}^{\mathsf{rem}} \subset \mathcal{E}$ and $\mathcal{E}^{\mathsf{add}} \subset (\mathcal{V}\times\mathcal{V})\setminus\mathcal{E}$. $\mathcal{E}^{\mathsf{Rew}}$ will be referred to as the rewired edge set.

We can extend the `MPNN` framework in Equation 2.1 to account for this rewiring:

$$\begin{aligned}\mathbf{z}_i^{(l)} = \mathsf{Upd}^{(l)} \Big(\mathbf{z}_i^{(l-1)}, \mathsf{Agg}^{(l)} \Big(\mathbf{z}_i^{(l-1)}, \Big\{ \mathbf{z}_j^{(l-1)} : j \in \mathcal{N}(i) \Big\} \Big),& \\ \mathsf{AggRew}^{(l)} \Big(\mathbf{z}_i^{(l-1)}, \Big\{ \mathbf{z}_j^{(l-1)} : j \in \mathcal{N}^{\mathsf{Rew}}(i) \Big\} \Big) \Big)&\end{aligned}$$

where $\mathcal{N}^{\mathsf{Rew}}(i)$ denotes the 1-hop neighborhood of node i in $\mathcal{E}^{\mathsf{Rew}}$. In a special case, which includes many of the rewiring techniques we will discuss, $\mathsf{Agg} = 0$, i.e. the original topology is discarded and only the rewired graph is used for message-passing.

Spatial rewiring methods use the topological relationships between the nodes in order to come up with a rewiring strategy. That is the graph rewiring is guided by the objective of optimizing some chosen topological properties. For instance, [AY21] introduced a *fully-adjacent* (FA) layer, wherein messages are passed between all nodes. `GNNs` using a FA layer in the final message-passing step were shown to outperform the baselines on a variety of long-range tasks, revealing the importance of information exchange between far off nodes which standard message-passing cannot facilitate. [Top+22] proposed a curvature-based rewiring strategy, called the Stochastic Discrete Ricci Flow (SDRF). It aims to reduce the "bottleneckedness" of a graph by adding suitable edges, while simultaneously removing edges in an effort to preserve the statistical properties of the original topology. [Bla+23] proposed the Greedy Total Resistance (GTR) technique, which optimizes the graph's total resistance (Definition 2.13) by greedily adding edges to achieve the greatest improvement (Theorem 2.2). One concern with graph rewiring methods is that unmoderated densification of the graph, e.g. using a fully connected graph for propagating messages, can result in a loss of the inductive bias the topology provides, potentially leading to over-fitting. Accordingly, [Gut+23] propose a Dynamically Rewired (DRew) message-passing framework that gradually *densifies* the graph. Specifically, in a given layer l, node i aggregates messages from its l-hop receptive field, $\mathcal{B}^{(l)}(i)$ (Definition 2.18), instead of the standard choice $\tilde{\mathcal{N}}(i) = \mathcal{B}^{(1)}(i)$. This results in an improved communication over long distances while also retaining the inductive bias of the shortest distance between nodes (Definition 2.5).

Spectral methods, on the other hand, use the spectral properties of the matrices encoding the graph topology, e.g. the adjacency matrix or the Laplacian matrix, to design rewiring algorithms. [Arn+22] propose a differentiable graph rewiring layer based on the

Lovász bound [Lov93, Corollary 3.3]:

$$\left|\mathbf{R}_{i,j} - \left(\frac{1}{d_i} + \frac{1}{d_j}\right)\right| \leq \frac{2}{\lambda_2 d_{\min}}$$

[Ban+22] introduced the Random Local Edge Flip (RLEF) algorithm, which draws inspiration from the "Flip Markov Chain" [MS05; Fed+06]. A sequence of such steps can convert a connected graph into an *expander graph* – a sparse graph with good connectivity (in terms of Cheeger's constant) – with high probability [MS05; Fed+06; All+16; Coo+19; Gia22], thereby enabling effective information propagation across the graph.

Some other rewiring techniques don't exactly classify as spatial or spectral methods. For instance, Probabilistically Rewired MPNN (PR-MPNN) [Qia+24] learns to probabilistically rewire a graph, effectively mitigating under-reaching as well as over-squashing. Finally, [GYS23] proposed connecting all nodes at most r-hops away, for some $r \in \mathbb{N}$, and introducing positional embeddings to allow for distance-aware aggregation of messages.

2.5.3 Towards a Unified Treatment

The problems of over-smoothing and over-squashing seem to be directly related, with the former arising when training deep GNNs and the latter when such deep GNNs are used for modelling LRIs. However, several studies have shown that an inevitable trade-off exists between the two, meaning that optimizing for one will compromise the other. For instance, [Ngu+23, Proposition 4.3] showed that positively curved edges[7] in a graph contribute towards the over-smoothing problem. Formally, if the all the edge curvatures in a graph are bounded below by a sufficiently high constant, then the Dirichlet energy of the node-level representations is expected to decay at an exponential rate. Consequently, graphs with a high number of positively curved edges are expected to suffer from over-smoothing. On the other hand, as discussed earlier, negatively curved edges create bottlenecks in the graph, resulting in over-squashing of information [Top+22; Ngu+23]. To address this trade-off between the two problems, Batch Ollivier-Ricci Flow (BORF) [Ngu+23] adds new edges adjacent to the negatively curved ones, and simultaneously removes positively curved ones. In a similar vein, [Gir+23] demonstrated that the minimum number of message-passing

[7] In terms of the Ollivier-Ricci Curvature [Oll09]

steps required to reach a given level of over-smoothing is inversely related to the Cheeger's constant, h_{G}. This again implies an inverse relationship between over-smoothing and over-squashing. To effectively alleviate the two issues together, [Gir+23] proposed the Stochastic Jost and Liu Curvature Rewiring (SJLR) algorithm. SJLR adds edges which result in high improvement in the curvature of existing edges, while simultaneously removing those that have low curvature.

Despite the well-established trade-off between over-smoothing and over-squashing, some works have successfully tackled them together despite *only* adding or removing edges. One such work is [KBM23], which proposed a rewiring algorithm that adds edges to the graph but does not remove any. The First-order Spectral Rewiring (FoSR) algorithm computes, as the name suggests, a first order approximation to the spectral gap of $\mathbf{L}^{\text{sym}}$, and adds edges with the aim of maximizing it. Since λ_2 directly relates to h_{G} through Cheeger's inequality, this directly decreases the over-squashing levels. Moreover, [KBM23, Figure 5] empirically demonstrated that addition of (up to a small number of) edges selected by FoSR can lower the Dirichlet energy of the representations. Taking a completely opposite approach, CurvDrop [Liu+23] adapted `DropEdge` to remove negatively curved edges sampled from a distribution proportional to their curvatures. CurvDrop directly reduces over-squashing and, as a side benefit of operating on a subgraph (Theorem 2.4), also mitigates over-smoothing.

2.5.4 Long Range Graph Benchmarks

Most research works, regardless of their objectives, perform evaluations on popular datasets, a good number of which are short-range tasks – either they are small world graphs (on average, $d_{\mathsf{G}}(i, j) \propto \log N$), they have high homophily (labels of nearby nodes are similar), or they are simply modelled better by shallower networks[8]. Examples of such datasets are citation networks [YCS16], social networks [LM12; HYL17] and product recommendations networks [McA+15; Shc+18]. On short-range tasks, deep models need not harness the full extent of their expressivity – they can simply fit to the signals local to each node. There-

[8]Admittedly, this is an unreliable judgement criteria since without having alleviated vanishing gradients, over-smoothing and over-squashing, we cannot conclude whether performance degradation with increasing depth is due to model-dataset misalignment, or simply due to high optimization error.

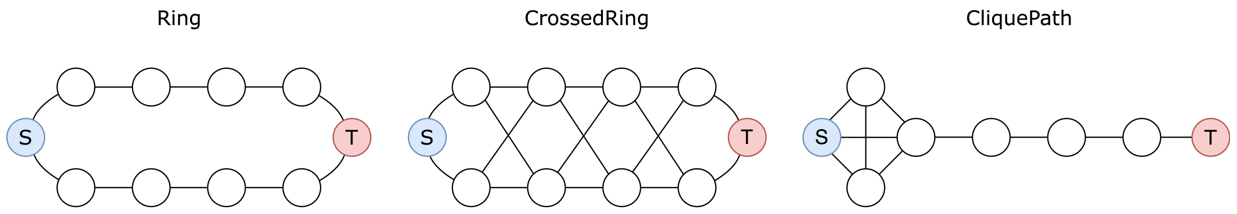


Figure 2.2: Topology of the graphs used in the `Ring`, `Crossed Ring` and `Clique Path Transfer` tasks. Figure from [Di +23].

fore, the ability to train deep `GNNs`, i.e. preventing loss of expressiveness as more layers are stacked, is not sufficient to conclude that they are capable of capturing LRIs. To test for that, we need to evaluate the algorithms on long-range datasets.

Unfortunately, it is not easy to conclude if a graph-learning tasks entails modelling of LRIs. Synthetic datasets help overcome this challenge by allowing us to manually control the range of interaction in the underlying ground-truth. [Mat+19] proposed 3 datasets to evaluate if `GNNs` can capture important LRIs in the graph. The first one was a task of finding the *unique path* connecting two nodes in tree structured graphs of varying sizes (up to a 100 nodes). The second was a maze solving task, which is arguably more difficult to solve. Specifically, some nodes in a grid were labelled as 'walls', and the task entailed identifying an un-blocked path connecting a source node to a target. Finally, they introduced circuit graphs, which were derived by labelling the edges as wires, resistors or batteries; the goal was to estimate the voltage at each node. [SGB20] proposed the `Shortest Path Prediction` task in randomly generated graphs. More recently, in an attempt to demonstrate the over-squashing phenomenon, [AY21] introduced the `Neighbors Match` problem, wherein a target node needs to be matched with one of several source nodes, such that their neighborhoods match. [Bod+21] introduced the `Ring Transfer` task, wherein a source and a target node are placed at diametrically opposite ends of a k-cycle, and the goal is to transfer the one-hot encoded label of the source to the target. [Di +23] extended the `Ring Transfer` task to other graph topologies – the `Crossed Ring` and the `Clique Path` – without changing any other settings; Figure 2.2 shows the topologies of the graphs used in these datasets. The `Color Connectivity` task proposed in [RW21] requires the model to predict whether a labelled subset containing 50% of all nodes forms one connected component or two disjoint ones. Since the tasks above are completely synthetic, the conclusions drawn from them remain somewhat unreliable. To add some level of realism to the tasks, [Gio+24] introduced `SyntheticZINC`, which uses molecular graphs from the ZINC chemical dataset [Irw+12]. The node features and the graph labels are set to control the level of mixing in the ground-truth; see Section 4.2.1 for more details.

Several real-world datasets have also been identified as suitable for testing the effectiveness of methods aimed at alleviating over-squashing (and, in general, learning LRIs). For instance, [Lim+21] propose 7 datasets for graph machine learning evaluation – YelpChi [Muk+21], Twitch-explicit [RAS21], deezer-europe [RS20], Facebook100 [TMP12], Pokec [LK14], ogbn-proteins and arXiv-year [Hu+20]. Reportedly, these datasets have low homophily scores and therefore, models need to learn more than just local information in order to perform well on them. The newly introduced Long Range Graph Benchmark [Dwi+23] includes 5 new large-scale graph datasets, which are derived from previously existing datasets: PascalVOC-SP, where PascalVOC refers to The PASCAL visual object classes challenge [Eve+10] and SP stands for super-pixels, COCO-SP [Lin+14], PCQM-Contact [Hu+21], Peptides-func and Peptides-struct [Sin+15]. The sheer scale of these datasets becomes evident when considering their statistics. For instance, COCO-SP contains 123.3K graphs with an *average* of 477 nodes and 2.7K edges, while PCQM-Contact features 529.4K molecular graphs with an average of 30 nodes and 61 edges. In contrast, the popular citation networks Cora [McC+00], CiteSeer [GBL98] and PubMed [Nam+12] have a *total* of 2.7K, 3.3K and 19.7K nodes, and 5.4K, 4.7K and 44.3K edges, respectively. These datasets provide a crucial benchmark for evaluating the performance of GNNs, particularly in scenarios where LRIs challenge traditional message-passing algorithms.

Chapter 3

Theory

In Section 2.5.2, we saw that most rewiring methods targeted at alleviating over-squashing add edges in order to reduce the bottleneckedness of the graph. Moreover, in Section 2.5.3, we noted a trade-off between over-squashing and over-smoothing. Interestingly, `DropEdge`, which was originally proposed to alleviate over-smoothing, only removes edges from the original topology. This raises concerns about the effects of `DropEdge` on over-squashing in `GNNs`.

Recall the intricate relationship between over-squashing and effective resistance, as discussed in Section 2.4.1 and Section 2.4.3. These results motivated us to study the effect of `DropEdge` on effective resistance in a computational graph. However, this is not straight-forward endeavour. To see that, first consider the case of one-shot `DropEdge` (Equation 2.7) – in this scenario, the expectation (over `DropEdge` masks) of the effective resistance is infinite because `DropEdge` disconnects any two nodes with a non-zero probability. This renders the measure uninformative. The same happens with the expected commute time. On the other hand, in the case of layer-wise `DropEdge` (Equation 2.8), the notion of expected effective resistance is meaningless. However, we *can* compute the expected commute time for a random walk, where edges are randomly dropped at each step; we call this the `DropEdge` *random walk*.

The rest of this chapter is organised as follows. In Section 3.1, we introduce the

DropEdge random walk and show that its expected commute times are larger than those of a regular random walk, suggesting that DropEdge might actually aggravate the over-squashing problem. We study this effect in detail in Section 3.2. Specifically, in Section 3.2.1 and Section 3.2.2, we present novel theoretical results characterizing the negative effects of DropEdge on sensitivity in linear GCNs. We then extend these findings to nonlinear MPNNs in Section 3.2.3, building on the analysis from Section 2.4.1. Lastly, in Section 3.2.4, we show that our conclusions also apply to Monte Carlo DropEdge.

3.1 DropEdge Random Walk

Consider a connected, undirected graph $\mathsf{G}\left(\mathcal{V},\mathcal{E}\right)$, with adjacency matrix $\mathbf{A}$, degree matrix $\mathbf{D}$ and (unnormalized) Laplacian matrix $\mathbf{L}$. The computation of expected commute time in a random walk (Theorem 2.3) has only one source of randomness – in the choice of the edge traversed at each step. Here we add another source of randomness to this walk by randomly dropping edges with probability q at each step of the walk; we call this the DropEdge(q) *random walk*. To account for the possibility that at iteration t, all the edges adjacent to s_t may be dropped, we assume that the walk simply stays at s_t for that iteration. Going forward, we assume $q \in [0, 1)$, since $q = 1$ leads to the degenerate case where the random walk fails to leave the start node, and all commute times between two different nodes are trivially equal to ∞.

Lemma 3.1. *The transition matrix of a DropEdge(q) random walk on* G *is given by*

$$\mathbf{P}^{(q)} = \left(\mathbf{I}_N - q^{\mathbf{D}}\right)\mathbf{D}^{-1}\mathbf{A} + q^{\mathbf{D}}\mathbf{I}_N$$

where $q^{\mathbf{D}} = \mathit{diag}\left(q^{d_1}, \ldots, q^{d_N}\right)$.

Proof. First, we derive the transition probabilities marginalizing out the randomness from DropEdge. If nodes i and j are connected, then

$$
\begin{aligned}
\mathbf{P}_{ij}^{(q)} &= q \cdot 0 + (1-q)\,\mathbb{E}\left[\frac{1}{1+\sum_{k=1}^{d_i-1} m_k}\right] \\
&= (1-q)\,\mathbb{E}\left[\frac{1}{1+M_i}\right] \\
&= (1-q)\sum_{k=0}^{d_i-1}\binom{d_i-1}{k}(1-q)^k\,(q)^{d_i-k-1}\left(\frac{1}{1+k}\right) \\
&= \frac{1}{d_i}\sum_{k=0}^{d_i-1}\binom{d_i}{k+1}(1-q)^{k+1}\,(q)^{d_i-k-1} \\
&= \frac{1}{d_i}\sum_{k=1}^{d_i}\binom{d_i}{k}(1-q)^k\,(q)^{d_i-k} \\
&= \frac{1-q^{d_i}}{d_i} \qquad (3.1)
\end{aligned}
$$

where $m_k \sim \operatorname{Bern}(1-q)$ and $M_i \sim \operatorname{Binom}(d_i-1, 1-q)$. If $i=j$, the random walk stays at node i with an additional probability q^{d_i} – the case where all edges adjacent to node i are dropped. This can be seen as a weighted sum of the original transition probabilities $\mathbf{P}_{ij}^{(0)} := \mathbf{A}_{ij}/d_i = \mathbf{L}_{ij}^{\text{asym}}$ and self-transitions $\mathbf{P}_{ij}^{S} = \delta_{ij}$, with weights $1-q^{d_i}$ and q^{d_i}, respectively, where δ denotes the Kronecker-delta. The transition matrix is given by

$$
\begin{aligned}
\mathbf{P}^{(q)} &= \left(\mathbf{I}_N - q^{\mathbf{D}}\right)\mathbf{P}^{(0)} \quad + q^{\mathbf{D}}\mathbf{P}^{S} \\
&= \left(\mathbf{I}_N - q^{\mathbf{D}}\right)\mathbf{D}^{-1}\mathbf{A} + q^{\mathbf{D}}\mathbf{I}_N
\end{aligned}
$$

□

Lemma 3.2. *The stationary distribution of the* `DropEdge`*(q) random walk on* G *is given by*

$$
\boldsymbol{\pi}^T = \frac{1}{\beta^{(q)}}\begin{bmatrix}\frac{d_1}{1-q^{d_1}} & \frac{d_2}{1-q^{d_2}} & \cdots & \frac{d_N}{1-q^{d_N}}\end{bmatrix}
$$

where the scaling factor $\beta^{(q)}$ *is given as*

$$
\beta^{(q)} = \sum_{i=1}^{N}\frac{d_i}{1-q^{d_i}} \qquad (3.2)
$$

Proof. We begin with the stationarity condition:

$$\begin{aligned}\boldsymbol{\pi}^T\mathbf{P}^{(q)} = \boldsymbol{\pi}^T \implies \mathbf{0}_N^T &= \boldsymbol{\pi}^T\left[\mathbf{I}_N - \mathbf{P}^{(q)}\right] \\ &= \boldsymbol{\pi}^T\left[\mathbf{I}_N - \left(\mathbf{P}^{(0)} + q^{\mathbf{D}}\left(\mathbf{I}_N - \mathbf{P}^{(0)}\right)\right)\right] \\ &= \boldsymbol{\pi}^T\left(\mathbf{I}_N - q^{\mathbf{D}}\right)\left(\mathbf{I}_N - \mathbf{P}^{(0)}\right)\end{aligned}$$

Since the column sums of the (in-degree) graph Laplacian are 0,

$$\begin{aligned}\mathbf{0}_N^T &= \mathbf{1}_N^T\mathbf{L} \\ &= \mathbf{1}_N^T\left(\mathbf{D} - \mathbf{A}\right) \\ &= \mathbf{1}_N^T\mathbf{D}\left(\mathbf{I}_N - \mathbf{P}^{(0)}\right) \\ &= \begin{bmatrix} d_1 & d_2 & \dots & d_N\end{bmatrix}\left(\mathbf{I}_N - \mathbf{P}^{(0)}\right)\end{aligned}$$

Since the graph is connected, there is only one left-eigenvector of $\mathbf{I}_N - \mathbf{P}^{(0)}$ corresponding to the eigenvalue 0. Hence,

$$\begin{aligned}\boldsymbol{\pi}^T\left(\mathbf{I}_N - q^{\mathbf{D}}\right) &= \frac{1}{\beta^{(q)}}\begin{bmatrix} d_1 & d_2 & \dots & d_N\end{bmatrix} \\ \implies \boldsymbol{\pi}^T &= \frac{1}{\beta^{(q)}}\begin{bmatrix} \frac{d_1}{1-q^{d_1}} & \frac{d_2}{1-q^{d_2}} & \dots & \frac{d_N}{1-q^{d_N}}\end{bmatrix}\end{aligned}$$

for some constant $\beta^{(q)} \in \mathbb{R}^+$. Since $\boldsymbol{\pi}$ denotes a probability distribution,

$$\beta^{(q)} = \sum_{i=1}^{N} \frac{d_i}{1 - q^{d_i}}$$

□

Theorem 3.1 (Mean First Passage Time)**.** *The expected first passage time (Definition 2.15) of a* `DropEdge`*(q) random walk on* G *is given by*

$$\mathbf{T}_{ij} = \beta^{(q)}\left[\left(\mathbf{L}_{jj}^{\dagger} - \mathbf{L}_{ij}^{\dagger}\right) - \sum_{k=1}^{N}\boldsymbol{\pi}_k\left(\mathbf{L}_{jk}^{\dagger} - \mathbf{L}_{ik}^{\dagger}\right)\right]$$

Proof. Using Definition 2.16, we have the mean return times given by

$$\mathbf{T}^+_{ij} = \mathbf{P}^{(q)}_{ij} \cdot 1 + \sum_{k \neq j} \mathbf{P}^{(q)}_{ik} \left(\mathbf{T}^+_{kj} + 1 \right) = 1 + \sum_{k \neq j} \mathbf{P}^{(q)}_{ik} \mathbf{T}^+_{kj}$$

where $\mathbf{P}^{(q)}$ denotes the transition matrix, as derived in Lemma 3.1. Let $\mathbf{E} = \mathbf{1}_{N \times N}$ and define $\mathbf{T}^+_d = \operatorname{diag}(\mathbf{T}^+)$. Then,

$$\mathbf{T}^+ = \mathbf{E} + \mathbf{P}^{(q)} \left(\mathbf{T}^+ - \mathbf{T}^+_d \right) \tag{3.3}$$

With the stationary distribution $\boldsymbol{\pi}$, satisfying $\boldsymbol{\pi}^T \mathbf{P}^{(q)} = \boldsymbol{\pi}^T$, we can derive T^+_d as

$$\begin{aligned} \boldsymbol{\pi}^T \mathbf{T}^+ &= \boldsymbol{\pi}^T \mathbf{1}_N \cdot \mathbf{1}_N^T + \boldsymbol{\pi}^T \mathbf{P}^{(q)} \left(\mathbf{T}^+ - \mathbf{T}^+_d \right) \\ &= \mathbf{1}_N^T + \boldsymbol{\pi}^T \mathbf{T}^+ - \boldsymbol{\pi}^T \mathbf{T}^+_d \\ \implies \mathbf{1}_N &= \mathbf{T}^+_d \boldsymbol{\pi} \\ \implies \mathbf{T}^+_{ii} &= \frac{1}{\boldsymbol{\pi}_i} \end{aligned}$$

Moreover, since the mean hitting times (Definition 2.15) satisfy $\mathbf{T} = \mathbf{T}^+ - \mathbf{T}^+_d$, Equation 3.3 becomes

$$\begin{aligned} \mathbf{T} + \mathbf{T}^+_d &= \mathbf{E} + \mathbf{P}^{(q)} \mathbf{T} \\ \implies \left(\mathbf{I}_N - \mathbf{P}^{(q)} \right) \mathbf{T} &= \mathbf{E} - \mathbf{T}^+_d \\ \implies \mathbf{D} \left(\mathbf{I}_N - \mathbf{P}^{(q)} \right) \mathbf{T} &= \mathbf{D} \left(\mathbf{E} - \mathbf{T}^+_d \right) \end{aligned} \tag{3.4}$$

Define $\mathbf{K} = \mathbf{D} \left(\mathbf{I}_N - \mathbf{P}^{(q)} \right)$. $\mathbf{K}$ relates to $\mathbf{L}$ as

$$\begin{aligned} \mathbf{K} &= \mathbf{D} - \mathbf{D}\mathbf{P}^{(q)} \\ &= \mathbf{D} - \left(\mathbf{I}_N - q^{\mathbf{D}} \right) \mathbf{A} - \mathbf{D} q^{\mathbf{D}} \\ &= \left(\mathbf{I}_N - q^{\mathbf{D}} \right) (\mathbf{D} - \mathbf{A}) = \left(\mathbf{I}_N - q^{\mathbf{D}} \right) \mathbf{L} \end{aligned}$$

It is easy to verify that $\mathbf{K}^\dagger = \mathbf{L}^\dagger \left(\mathbf{I}_N - q^{\mathbf{D}} \right)^{-1}$. Left-multiplying Equation 3.4 by $\mathbf{K}^\dagger$ we get

$$\mathbf{K}^\dagger \mathbf{D} \left(\mathbf{E} - \mathbf{T}^+_d \right) = \mathbf{K}^\dagger \mathbf{K} \mathbf{T} = \mathbf{T} - \mathbf{1}_N \mathbf{u}^T \implies \mathbf{T} = \mathbf{K}^\dagger \mathbf{D} \mathbf{E} - \mathbf{K}^\dagger \mathbf{D} \mathbf{T}^+_d + \mathbf{1}_N \mathbf{u}^T$$

where $\mathbf{u}^T = 1/N \cdot \mathbf{1}_N^T \mathbf{T}$ is proportional to the column sums of $\mathbf{T}$. This means that

$$\mathbf{T}_{ij} = \sum_{k=1}^{N} \mathbf{K}^\dagger_{ik} d_k - \mathbf{K}^\dagger_{ij} d_j \cdot \frac{1}{\boldsymbol{\pi}_j} + \mathbf{u}_j \tag{3.5}$$

Since the diagonal of $\mathbf{T}$ is 0,

$$\mathbf{u}_i = -\sum_{k=1}^{N} \mathbf{K}^\dagger_{ik} d_k + \mathbf{K}^\dagger_{ii} d_i \cdot \frac{1}{\boldsymbol{\pi}_i}$$

Plugging this back into Equation 3.5,

$$\begin{aligned}\mathbf{T}_{ij} &= \sum_{k=1}^{N} \mathbf{K}^\dagger_{ik} d_k - \mathbf{K}^\dagger_{ij} d_j \cdot \frac{1}{\boldsymbol{\pi}_j} - \sum_{k=1}^{N} \mathbf{K}^\dagger_{jk} d_k + \mathbf{K}^\dagger_{jj} d_j \cdot \frac{1}{\boldsymbol{\pi}_j} \\ &= \left(\mathbf{K}^\dagger_{jj} - \mathbf{K}^\dagger_{ij}\right) \frac{d_j}{\boldsymbol{\pi}_j} - \sum_{k=1}^{N} \left(\mathbf{K}^\dagger_{jk} - \mathbf{K}^\dagger_{ik}\right) d_k\end{aligned}$$

Finally, substituting $\mathbf{K}^\dagger = \mathbf{L}^\dagger \left(\mathbf{I}_N - q^{\mathbf{D}}\right)^{-1}$ and using Lemma 3.2, we conclude

$$\mathbf{T}_{ij} = \beta^{(q)} \left[\left(\mathbf{L}^\dagger_{jj} - \mathbf{L}^\dagger_{ij}\right) - \sum_{k=1}^{N} \boldsymbol{\pi}_k \left(\mathbf{L}^\dagger_{jk} - \mathbf{L}^\dagger_{ik}\right)\right]$$

□

Corollary 3.1 (`DropEdge` Commute Time)**.** *The expected commute time of a* `DropEdge`*(q) random walk on* G *is given by*

$$\mathbf{C}^{(q)}_{ij} = \beta^{(q)} \left(\mathbf{L}^\dagger_{ii} + \mathbf{L}^\dagger_{jj} - 2\mathbf{L}^\dagger_{ij}\right)$$

Proof. This is an immediate consequence of Definition 2.17 and Theorem 3.1. □

Note that if $q = 0$, we recover Theorem 2.3. Moreover, in the limit $q \to 1$, we have $\mathbf{C}^{(q)}_{ij} \to \infty$ for $i \neq j$, as expected.

Corollary 3.1 suggests that `DropEdge` *scales up the commute times* between every pair of nodes, by a factor which depends only on the degree distribution of the graph and the

`DropEdge` probability. One might think that it is obvious for $\mathbf{C}_{ij}$ to increase given that removing any edge increases $\mathbf{R}_{ij}$ (Theorem 2.2) and that the two quantities are proportional (Theorem 2.3), $\mathbf{C}_{ij} = \mathsf{vol}(\mathsf{G})\,\mathbf{R}_{ij}$. However, removing edges also decreases the proportionality constant, $\mathsf{vol}(\mathsf{G}) = 2\,|\mathcal{E}|$. Therefore, the pairwise commute times need not increase when edges are removed. See Figure 4.1, for example – graph 2 is a connected subgraph of graph 1, yet all the commute times from the source (node 0) are lower in graph 2 than in graph 1. Corollary 3.1 does *not* suggest that commute times in subgraphs are higher, but instead that if i.i.d. subgraphs are used in each step of the walk (cf. layer-wise `DropEdge` in Equation 2.8), then the expected commute times increase.

3.1.1 Inspecting the Scaling Factor

Let's take a closer look at the individual terms in Equation 3.2 and make a few observations. Start by setting

$$\beta_i^{(q)} = \frac{d_i}{1 - q^{d_i}}$$

1. $\beta_i^{(q)}$ is an increasing function of q. That is, the contribution to $\beta^{(q)}$ coming from each node monotonically increases as `DropEdge` probability is increased. Precisely speaking, for $q \in (0,1)$, the partial derivative w.r.t. q is given by

$$\frac{\partial \beta_i^{(q)}}{\partial q} = \left(\beta_i^{(q)}\right)^2 q^{d_i - 1} > 0$$

2. Similarly, the second derivative w.r.t. q is given by

$$\frac{\partial^2 \beta_i^{(q)}}{\partial q^2} = 2\left(\beta_i^{(q)}\right)^3 \left(q^{d_i-1}\right)^2 + (d_i - 1)\left(\beta_i^{(q)}\right)^2 q^{d_i-2} > 0$$

   This implies that the commute times become increasingly sensitive to the `DropEdge` probability as it is increased from 0 to 1.

3. The relative change w.r.t. a `NoDrop` baseline ($q = 0$) is lower for higher d_i:

$$\frac{\beta_i^{(q)}}{\beta_i^{(0)}} = \frac{1}{1 - q^{d_i}} \implies \frac{\partial}{\partial d_i}\left(\frac{\beta_i^{(q)}}{\beta_i^{(0)}}\right) = \frac{q^{d_i} \ln q}{\left(1 - q^{d_i}\right)^2} < 0$$

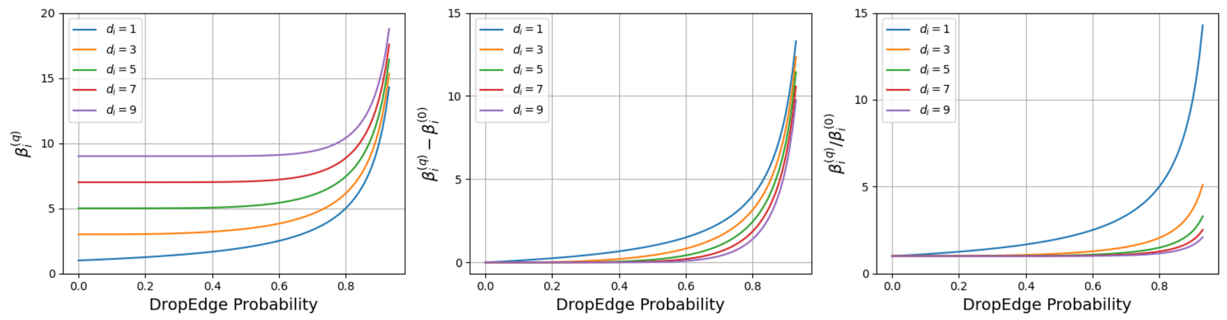


Figure 3.1: The effect of node degree, d_i, and `DropEdge` probability, q, on the individual terms in Equation 3.2.

Note that the relative change in commute times, $\mathbf{C}_{ij}^{(q)}/\mathbf{C}_{ij}^{(0)}$, is proportional to $\beta^{(q)}/\beta^{(0)}$. This means that `DropEdge` has a smaller effect on commute times in graphs with better connectivity (in the sense of having lesser nodes with low degree).

4. Similarly, the absolute change is also lower for higher d_i:

$$\beta_i^{(q)} - \beta_i^{(0)} = d_i \left(\frac{1}{1 - q^{d_i}} - 1 \right)$$

$$\implies \frac{\partial}{\partial d_i} \left(\beta_i^{(q)} - \beta_i^{(0)} \right) = \frac{q^{d_i}}{1 - q^{d_i}} + \frac{d_i q^{d_i} \ln q}{\left(1 - q^{d_i}\right)^2} = \frac{q^{d_i} \left(1 + \ln q^{d_i} - q^{d_i}\right)}{\left(1 - q^{d_i}\right)^2} < 0$$

This intuitively makes sense since the probability of the random walk transitioning out of a node (i.e. not having all edges adjacent to it being dropped) is higher when the node's degree is high.

Figure 3.1 shows how $\beta_i^{(q)}$, $\beta_i^{(q)} - \beta_i^{(0)}$ and $\beta_i^{(q)}/\beta_i^{(0)}$ vary as a function of the `DropEdge` probability, for different values of d_i.

Directly analyzing the effect of the degree sequence $\{d_1, \dots, d_N\}$ on $\beta^{(q)}$ is not as straight forward. Therefore, we do so empirically using the molecular graphs in the `Proteins` [DD03] and `MUTAG` [KMB05] datasets. Figure 3.2 shows the distributions of the ratio of scaling factors $\beta_i^{(q)}/\beta_i^{(0)}$. As expected, the distribution shifts to the right as the `DropEdge` probability is increased, implying that more and more graphs are expected to significantly suffer from increasing commute times.

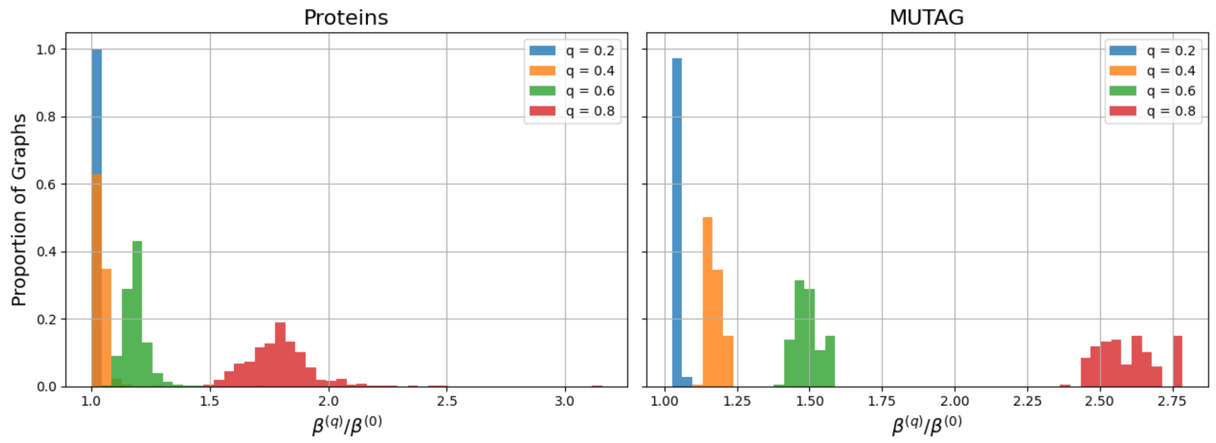


Figure 3.2: Distribution of the scaling factor $\beta^{(q)}/\beta^{(0)}$ over the `Proteins` dataset.

3.2 Sensitivity in a DropEdge Model

In Section 3.1, we studied the effect of `DropEdge` on commute times, noting that a high `DropEdge` probability increases them between all node pairs. Now, we study its effects on the sensitivity of one node's representations w.r.t. another node's input features. For this experiment, we use randomly initialized 6-layer `GCNs` with each layer's width set to 64, and `ReLU` activation after each message passing step. Molecular graphs are sampled from the `Proteins` and the `MUTAG` datasets, discarding those that are disconnected, and the following quantities are computed:

1. the pairwise commute distances $\mathbf{C}_{ij}$, as in Theorem 2.3, and
2. the pairwise sensitivities, $\left\|\partial \mathbf{z}_i^{(6)}/\partial \mathbf{x}_j\right\|_1$, where $\mathbf{z}_i^{(6)}$ represents the representation of node i in the 6th layer and $\mathbf{x}_j$ represents the input features of node j.

The node pairs (i,j) are binned by rounding their corresponding commute times to the nearest multiple of 40, and the mean of the sensitivity is computed for each bin. Figure 3.3 shows the sensitivity values against the (binned) commute times. First off, we observe an *exponential rate of decay in sensitivity* as distance between the nodes i and j increases, in agreement with [AY21; Top+22]. Secondly, `DropEdge` increases the decay rate, so that the sensitivity between nodes with high commute times is reduced, as expected

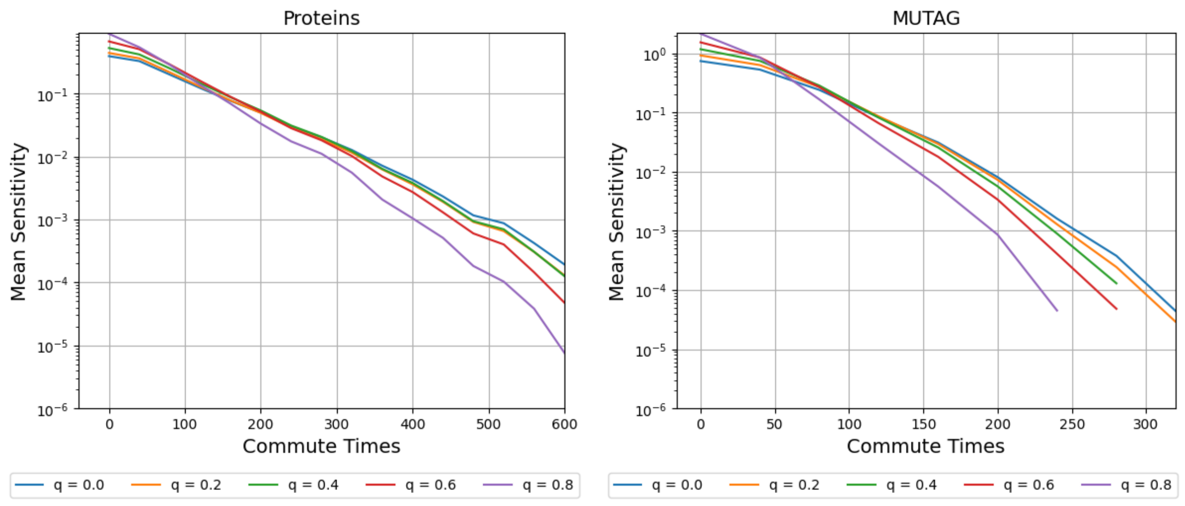


Figure 3.3: Sensitivity versus commute times in `Proteins` and `MUTAG` datasets. The error bars corresponds to one standard deviation w.r.t. the initialization of the `GCN`.

from Corollary 3.1. Note that this reduced sensitivity between nodes with high commute times may be beneficial in some cases. For example, when the task entails learning only short-range interactions, high `DropEdge` probability can decrease harmful redundancy in the model [Di +23]. Unexpectedly, `DropEdge` also seems to *increase the sensitivity between node pairs with small expected commute times.* Note that this does not contradict the results in [Di +23; Gio+24], since those results gave *bounds* on over-squashing levels, and studying the effect of `DropEdge` on commute times cannot precisely predict the effect it would have on sensitivity in `GNNs`.

3.2.1 1-Layer Linear GCN

To make sense of the observations in Figure 3.3, we perform a theoretical analysis of the expectation – w.r.t. `DropEdge` – of sensitivity in a linear `GCN`. Consider an input $(\mathsf{G}(\mathcal{V},\mathcal{E}), \mathbf{X})$, where $\mathbf{X} \in \mathbb{R}^{N\times H^{(0)}}$. As a starting point, we consider a 1-layer linear `GCN` with an asymmetrically normalized propagation matrix, $\hat{\mathbf{A}}^{\text{asym}}$ (Equation 2.4). Specifically, the individual feature vectors are given by

$$\begin{aligned}
\mathbf{M}^{(1)} &\sim \{\text{Bern}\,(1-q)\}^{N\times N} \\
\tilde{\mathbf{A}}^{(1)} &= \mathbf{M}^{(1)} \circ \mathbf{A} + \mathbf{I}_N \\
\tilde{\mathbf{D}}^{(1)} &= \text{diag}\left(\tilde{\mathbf{A}}^{(1)}\mathbf{1}_N\right) \\
\hat{\mathbf{A}}^{(1)} &= \left(\tilde{\mathbf{D}}^{(1)}\right)^{-1}\tilde{\mathbf{A}} \\
\mathbf{z}_i^{(1)} &= \sum_{j=1}^{N}\hat{\mathbf{A}}_{ij}^{(1)}\mathbf{x}_j\mathbf{W}^{(1)}
\end{aligned} \tag{3.6}$$

Theorem 3.2. *In the case of a 1-layer linear* `GCN` *with* $\hat{\mathbf{A}} = \hat{\mathbf{A}}^{asym}$*, using* `DropEdge`

1. *increases the sensitivity of a node's representations to its own input features, and*
2. *decreases the sensitivity to its neighbors' features.*

Proof. Since derivatives and matrices are linear operators, the Jacobian of $\mathbf{z}_i^{(1)}$ w.r.t. $\mathbf{x}_j$ is

$$\frac{\partial \mathbf{z}_i^{(1)}}{\partial \mathbf{x}_j} = \hat{\mathbf{A}}_{ij}^{(1)}\mathbf{W}^{(1)}$$

Sensitivity is given by the 1-norm of the Jacobian (Definition 2.23). In a `NoDrop` model ($q = 0$), this would simply be given by

$$\left\|\frac{\partial \mathbf{z}_i^{(1)}}{\partial \mathbf{x}_j}\right\|_1 = \hat{\mathbf{A}}_{ij}^{(1)}\left\|\mathbf{W}^{(1)}\right\|_1 = \frac{1}{d_i+1}\left\|\mathbf{W}^{(1)}\right\|_1$$

where d_i is the *in-degree* of node i. In a `DropEdge` model ($q > 0$), we are interested in the expected sensitivity, marginalizing out the randomness from `DropEdge`:

$$\mathbb{E}_{\mathbf{M}^{(1)}\sim\{\text{Bern}(1-q)\}^{N\times N}}\left[\left\|\frac{\partial \mathbf{z}_i^{(1)}}{\partial \mathbf{x}_j}\right\|_1\right] = \mathbb{E}_{\mathbf{M}^{(1)}}\left[\hat{\mathbf{A}}_{ij}^{(1)}\right]\left\|\mathbf{W}^{(1)}\right\|_1$$

Since we are not considering the effect of `DropEdge` on the learning trajectory, we can ignore the weight matrix and simply compare the respective entries in the (expected) propagation matrix. Let's look at $\mathbb{E}_{\mathbf{M}^{(1)}}\left[\hat{\mathbf{A}}_{ii}^{(1)}\right]$, first. Recall that a self-loop is added to the graph *after* the edges are dropped, and then the (asymmetric) normalization is performed. In other words, the self-loop is never dropped. Following Lemma 3.1, we have

$$\begin{aligned}\mathbb{E}_{\mathbf{M}^{(1)}}\left[\hat{\mathbf{A}}_{ii}^{(1)}\right] &= \mathbb{E}_{m_1,\ldots,m_{d_i}}\left[\frac{1}{1+\sum_{k=1}^{d_i} m_k}\right] \\ &= \frac{1-q^{d_i+1}}{(1-q)\,(d_i+1)} \\ &> \frac{1}{d_i+1}\end{aligned} \tag{3.7}$$

This proves the first part of the theorem. On the other hand,

$$\begin{aligned}\mathbb{E}_{\mathbf{M}^{(1)}}\left[\hat{\mathbf{A}}_{ij}^{(1)}\right] &= (1-q)\,\mathbb{E}_{m_1,\ldots,m_{d_i-1}}\left[\frac{1}{2+\sum_{k=1}^{d_i-1} m_k}\right] \\ &= (1-q)\sum_{k=0}^{d_i-1}\binom{d_i-1}{k}(1-q)^k\,(q)^{d_i-k-1}\left(\frac{1}{2+k}\right) \\ &= \sum_{k=0}^{d_i-1}\frac{(d_i-1)!}{(k+2)!\,(d_i-k-1)!}(1-q)^{k+1}\,(q)^{d_i-k-1}\,(k+1) \\ &= \sum_{k=2}^{d_i+1}\frac{(d_i-1)!}{(k)!\,(d_i-k+1)!}(1-q)^{k-1}\,(q)^{d_i-k+1}\,(k-1) \\ &= \frac{1}{d_i\,(d_i+1)\,(1-q)}\sum_{k=2}^{d_i+1}\binom{d_i+1}{k}(1-q)^k\,(q)^{d_i-k+1}\,(k-1) \\ &= \frac{1}{d_i\,(d_i+1)\,(1-q)}\left[(d_i+1)\,(1-q)-q^{d_i+1}\right] \\ &= \frac{1}{d_i}\left(1-\mathbb{E}_{\mathbf{M}^{(1)}}\left[\hat{\mathbf{A}}_{ii}^{(1)}\right]\right) \\ &< \frac{1}{d_i+1}\end{aligned} \tag{3.8}$$

This proves the second part of the theorem.

$\square$

We conclude that `DropEdge` reduces the sensitivity of a node's representation to its 1-hop neighbors' features. In fact, in the limit of $q \to 1$, i.e. when all the edges are *almost surely* dropped, the `GCN` completely ignores the underlying topology and the sensitivity between the neighbors disappears, reducing the model to an `MLP`, as expected.

3.2.2 L-Layer Linear GCN

Now, consider an L-layer `GCN` in which `DropEdge` masks are sampled identically and independently in each layer (as is commonly done in `GNN` training):

$$\begin{aligned}
\mathbf{M}^{(l)} &\overset{\text{iid}}{\sim} \{\text{Bern}\,(1-q)\}^{N\times N} \\
\tilde{\mathbf{A}}^{(l)} &= \mathbf{M}^{(l)} \circ \mathbf{A} + \mathbf{I}_N \\
\tilde{\mathbf{D}}^{(l)} &= \text{diag}\left(\tilde{\mathbf{A}}^{(l)}\mathbf{1}_N\right) \\
\hat{\mathbf{A}}^{(l)} &= \left(\tilde{\mathbf{D}}^{(l)}\right)^{-1}\tilde{\mathbf{A}}^{(l)} \\
\mathbf{Z}^{(l)} &= \hat{\mathbf{A}}^{(l)}\mathbf{Z}^{(l-1)}\mathbf{W}^{(l)}
\end{aligned} \tag{3.9}$$

Then, the final node representations are given by

$$\mathbf{Z}^{(L)} = \bar{\mathbf{A}}\mathbf{X}\bar{\mathbf{W}} \in \mathbb{R}^{N\times H^{(L)}}$$

where $\bar{\mathbf{A}} \coloneqq \prod_{l=1}^{L}\hat{\mathbf{A}}^{(l)} \in \mathbb{R}^{N\times N}$ and $\bar{\mathbf{W}} \coloneqq \prod_{l=1}^{L}\mathbf{W}^{(l)} \in \mathbb{R}^{H^{(0)}\times H^{(L)}}$.

Lemma 3.3. *The expected sensitivity between nodes i and j in an L-layer linear `GCN` with `DropEdge` probability q is*

$$\mathbb{E}_{\mathbf{M}^{(1)},\ldots,\mathbf{M}^{(L)}}\left[\left\|\frac{\partial \mathbf{z}_i^{(L)}}{\partial \mathbf{x}_j}\right\|_1\right] = \left(\dot{\mathbf{P}}^L\right)_{ij}\left\|\bar{W}\right\|_1$$

where $\dot{\mathbf{P}} \coloneqq \mathbb{E}_{\mathbf{M}}\left[\hat{\mathbf{A}}^{(l)}\right]$ is the expected propagation matrix.

Proof.

$$\mathbf{z}_i^{(L)} = \sum_{j=1}^{N} \bar{\mathbf{A}}_{ij}\mathbf{x}_j\bar{W} \implies \frac{\partial \mathbf{z}_i^{(L)}}{\partial \mathbf{x}_j} = \bar{\mathbf{A}}_{ij}\bar{W} \implies \left\| \frac{\partial \mathbf{z}_i^{(L)}}{\partial \mathbf{x}_j} \right\|_1 = \bar{\mathbf{A}}_{ij}\left\|\bar{W}\right\|_1$$
$$\implies \mathbb{E}_{\mathbf{M}^{(1)},\ldots,\mathbf{M}^{(L)}}\left[\left\| \frac{\partial \mathbf{z}_i^{(L)}}{\partial \mathbf{x}_j} \right\|_1\right] = \mathbb{E}_{\mathbf{M}^{(1)},\ldots,\mathbf{M}^{(L)}}\left[\bar{\mathbf{A}}_{ij}\right]\left\|\bar{W}\right\|_1$$

Using the i.i.d. assumption on the distribution of `DropEdge` masks, the expectation of $\bar{\mathbf{A}}$ is given by

$$\mathbb{E}_{\mathbf{M}^{(1)},\ldots,\mathbf{M}^{(L)}}\left[\bar{\mathbf{A}}\right] = \mathbb{E}_{\mathbf{M}^{(1)},\ldots,\mathbf{M}^{(L)}}\left[\prod_{l=1}^{L}\hat{\mathbf{A}}^{(l)}\right] = \left(\mathbb{E}_{\mathbf{M}}\left[\hat{\mathbf{A}}^{(l)}\right]\right)^L = \dot{\mathbf{P}}^L$$

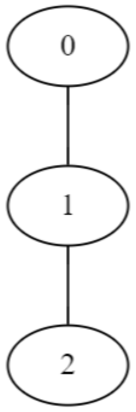
which concludes the proof. $\square$

Note that $\dot{\mathbf{P}}$ is the transition matrix of a `DropEdge` random walk on the augmented graph (Definition 2.24), and $\left(\dot{\mathbf{P}}^L\right)_{ij}$ denotes the total probability of transitioning from node j to node i in exactly L steps. As a sanity check, we consider the limit $q \to 1$. In this case $\dot{\mathbf{P}}, \dot{\mathbf{P}}^L \to \mathbf{I}_N$ which means: 1. $\left(\dot{\mathbf{P}}^L\right)_{ij} \to 0 \implies \left\|\partial \mathbf{z}_i^{(L)}/\partial \mathbf{x}_j\right\|_1 \to 0$ for $i \neq j$, that is the sensitivity of a node's representation to other nodes' features vanishes, and 2. $\left(\dot{\mathbf{P}}^L\right)_{ii} \to 1 > 1/\left(d_i+1\right)$, that is the sensitivity to its own features increases.

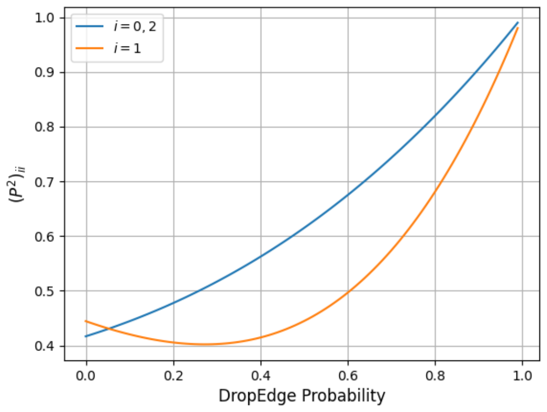
Now, let's consider the general case of $q \in (0,1)$. We recall the expressions for the entries of $\dot{\mathbf{P}}$ (derived in Equation 3.7 and Equation 3.8):

$$\dot{\mathbf{P}}_{mm} = \frac{1-q^{d_m+1}}{(1-q)\left(d_m+1\right)} > \frac{1}{d_m+1}$$
$$\dot{\mathbf{P}}_{mn} = \frac{1}{d_m}\left(1 - \frac{1-q^{d_m+1}}{(1-q)\left(d_m+1\right)}\right) < \frac{1}{d_m+1}$$

That is, compared to a `NoDrop` model, the self-transition probabilities increase while the probabilities of transitioning out of a state decrease. As a result, the probability of transitioning from one node to another in exactly L steps may increase or decrease, depending on the in-degrees of the nodes on the paths connecting them. Without making assumptions about the topology of the graph, we cannot make statements about the effect



(a) 3-node path graph



(b) Diagonal entries of $\dot{\mathbf{P}}^2$ versus q

Figure 3.4: Illustrating the dependence of propagation matrix entries on the topology and `DropEdge` probability using a 3-chain graph

`DropEdge` has on the sensitivity of a node's representations w.r.t. its own features (as we did for a 1-layer `GCN` in Theorem 3.2). For example, consider a 2-layer linear `GCN` and a chain graph with 3 nodes, as shown in Figure 3.4a. The diagonal entries of $\dot{\mathbf{P}}^2$ for this graph are shown in Figure 3.4b. While the entries corresponding to nodes 0 and 2 monotonically increase, as was expected for a 1-layer `GCN`, the diagonal entry corresponding to node 1 drops up until $q \approx 0.27$, before increasing.

A similar reasoning follows for nodes up to $(L-1)$ hops away. For nodes L hops away, however, we can show that `DropEdge` always decreases the corresponding entry in $\dot{\mathbf{P}}^L$, reducing the effective reachability of `GCNs`:

Theorem 3.3. *In the case of an L-layer linear `GCN` with the asymmetric propagation rule (Equation 3.9), using `DropEdge` decreases the sensitivity of a node $i \in \mathcal{V}$ to another node $j \in \mathcal{S}^{(L)}(i)$. Moreover, the sensitivity monotonically decreases as the `DropEdge` probability q is increased.*

Proof. Intuitively, since there is no self-loop on any given L-length path connecting nodes i and j (which are assumed to be L-hops away), the probability of each transition on any path connecting these nodes is reduced. Therefore, so is the total probability of transitioning from j to i in exactly L hops. More formally, denote the set of paths connecting them by

$$\mathsf{Paths}\,(i,j) = \{(u_0,\ldots,u_L) : u_0 = j; u_L = i; (u_{l-1},u_l) \in \mathcal{E}, \forall l \in [L]\}$$

The (i,j)-entry in the propagation matrix is given by

$$\left(\dot{\mathbf{P}}^L\right)_{ij} = \sum_{(u_0,\ldots,u_L)\in \text{Paths}(i,j)} \prod_{l=1}^{L} \dot{\mathbf{P}}_{u_{l-1}u_l} \tag{3.10}$$

Since there is no self-loop on any of these paths,

$$\begin{aligned}\left(\dot{\mathbf{P}}^L\right)_{ij} &= \sum_{(u_0,\ldots,u_L)\in \mathsf{Paths}(i,j)} \prod_{l=1}^{L} \frac{1}{d_{u_{l-1}}}\left(1 - \frac{1-q^{d_{u_{l-1}}+1}}{(1-q)\left(d_{u_{l-1}}+1\right)}\right) \\ &< \sum_{(u_0,\ldots,u_L)\in \mathsf{Paths}(i,j)} \prod_{l=1}^{L} \left(\frac{1}{d_{u_{l-1}}+1}\right)\end{aligned}$$

The right hand side of the inequality is the (i,j)-entry in the L^{th} power of the propagation matrix of a `NoDrop` model. We conclude the first part of the proof using Lemma 3.3 – the sensitivity of node i to node j is proportional to $\left(\dot{\mathbf{P}}^L\right)_{ij}$. Next, we recall the geometric series for any q:

$$1 + q + \ldots + q^d = \frac{1-q^{d+1}}{1-q}$$

Each of the terms on the right are increasing in q, hence, $\dot{\mathbf{P}}_{mn}$ is decreasing in q. Using this result with Equation 3.10, we conclude the second part of the theorem. □

As discussed earlier, the change in sensitivity between nodes up to $L-1$ hops away depends on the graph topology, and no general statements can be made about it. However, we can analyze this empirically. We start by sampling 100 molecular graphs from each of the `Proteins` and `MUTAG` datasets. For each graph, we compute its propagation matrix $\dot{\mathbf{P}}^L$, with $L=6$, bin the node pairs (i,j) by the shortest distance between them, and finally compute the average of the entries $\left(\dot{\mathbf{P}}^L\right)_{ij}$ in each of the bins. Finally, we average these bin-means over the 100 graphs. The results are shown in Figure 3.5. We observe that `DropEdge` increases the expected sensitivity between nodes close to each other (0-hop and 1-hop neighbors) in the original topology, but reduces it between nodes farther off.

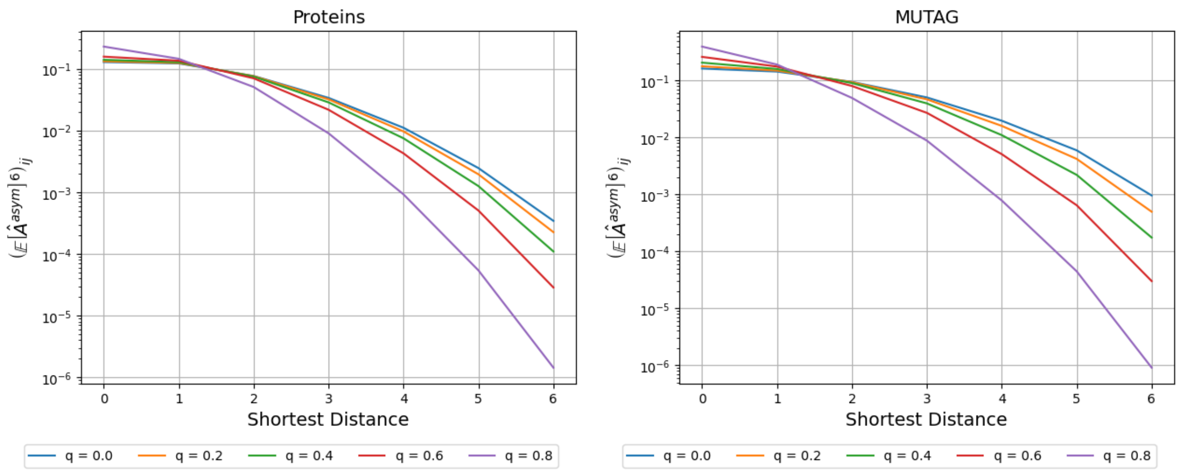


Figure 3.5: Entries of $\dot{\mathbf{P}}^6$, averaged after binning node-pairs by their shortest distance.

Note that these results correspond to a linear GCN using asymmetric normalization of the adjacency matrix for aggregating messages (Equation 2.4), i.e. in each message passing step, only the in-degree of node i is used to compute the weight of the message from the neighboring node j to target node i. In practice, however, it is more common to use symmetric normalization (Equation 2.3) instead.

$$\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$$

As in the case of Lemma 3.3, we are looking for

$$\mathbb{E}_{\mathbf{M}^{(1)},\ldots,\mathbf{M}^{(L)}}\left[\prod_{l=1}^{L}\hat{\mathbf{A}}^{(l)}\right] = \left(\mathbb{E}_{\mathbf{M}}\left[\hat{\mathbf{A}}^{\text{sym}}\right]\right)^{L}$$

While $\ddot{\mathbf{P}} := \mathbb{E}_{\mathbf{M}}\left[\hat{\mathbf{A}}^{\text{sym}}_{ij}\right]$ does not have a closed form expression, we can approximate this expectation using Monte-Carlo sampling. We use 10 samples of $\mathbf{M}$ to compute an approximation of $\ddot{\mathbf{P}}$, and plot out the average of its entries, as we did with $\dot{\mathbf{P}}$ in Figure 3.5. The results are presented in Figure 3.6, which shows that while the sensitivity between nearby nodes is affected to a lesser extent compared to the results in Figure 3.5, that between far-off nodes is significantly reduced, same as earlier. This result corroborates the findings in Figure 3.3, which correspond to a 6-layer *non-linear* GCN with symmetric normalization.

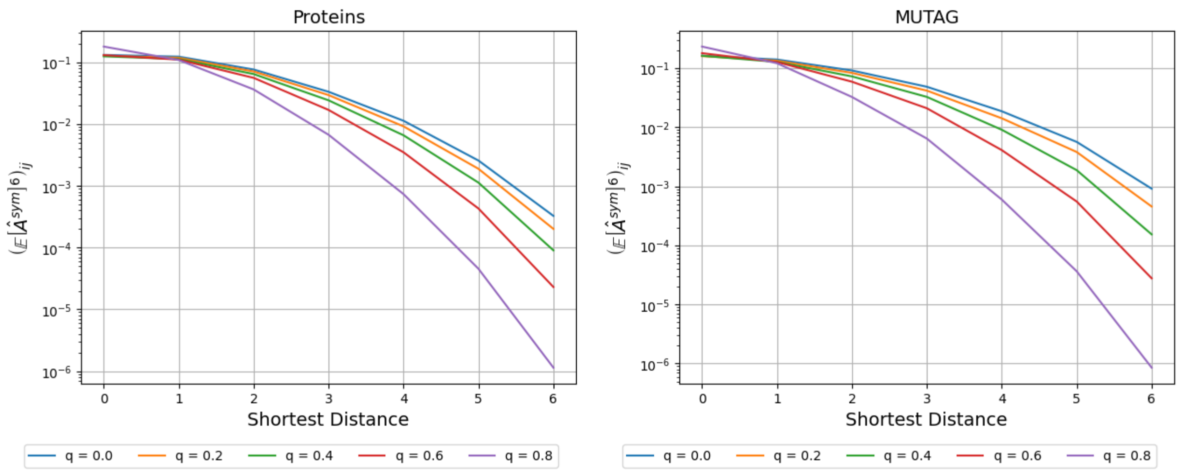


Figure 3.6: Entries of $\ddot{\mathbf{P}}^6$, averaged after binning node-pairs by their shortest distance.

3.2.3 L-layer Nonlinear MPNN

While linear networks are useful in simplifying the theoretical analysis, they are often not practical. In this section, we will extend the inequality bounds from Section 2.4.1 to the `DropEdge` setting.

Lemma 2.3 states that for a certain class of (non-linear) `MPNNs`[1] – which includes `GCNs` – the sensitivity of the model can be bounded as

$$\left\| \frac{\partial \mathbf{z}_i^{(L)}}{\partial \mathbf{x}_j} \right\|_1 \leq \zeta_1^L \left(\hat{\mathbf{A}}^L \right)_{ij}$$

where ζ is independent of the choice of nodes i and $j \in \mathcal{S}^{(L)}(i)$. Taking an expectation w.r.t. `DropEdge` on both sides of the inequality, we get

$$\mathbb{E}_{\mathbf{M}^{(1)},\ldots,\mathbf{M}^{(L)}} \left[\left\| \frac{\partial \mathbf{z}_i^{(L)}}{\partial \mathbf{x}_j} \right\|_1 \right] \leq \zeta_1^L \left(\mathbb{E}_{\mathbf{M}^{(1)},\ldots,\mathbf{M}^{(L)}} \left[\hat{\mathbf{A}}^L \right] \right)_{ij} = \zeta_1^L \left(\left(\mathbb{E}_{\mathbf{M}} \left[\hat{\mathbf{A}} \right] \right)^L \right)_{ij} \tag{3.11}$$

Theorem 3.3 tells us that $(\dot{\mathbf{P}}^L)_{ij}$ decreases monotonically with increasing `DropEdge` probability q. This implies that, in a non-linear `MPNN` with $\hat{\mathbf{A}} = \hat{\mathbf{A}}^{\text{asym}}$, `DropEdge` lowers the

[1]While Lemma 2.3 assumes $\hat{\mathbf{A}} = \hat{\mathbf{A}}^{\text{sym}}$, it does not use that in its proof (see [Top+22, Appendix A]); the lemma is valid for $\hat{\mathbf{A}} = \hat{\mathbf{A}}^{\text{asym}}$ as well.

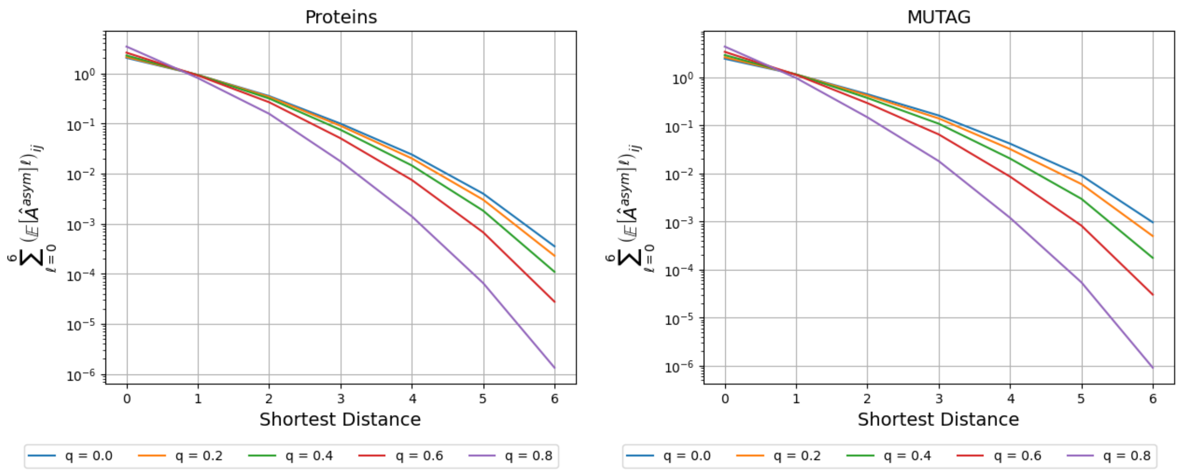


Figure 3.7: Entries of $\sum_{l=0}^{6} \dot{\mathbf{P}}^l$, averaged after binning node-pairs by their shortest distance.

sensitivity bound given above. Empirical results in Figure 3.6 support the same conclusion for $\hat{\mathbf{A}} = \hat{\mathbf{A}}^{\text{sym}}$.

Similarly, Lemma 2.4 bounds the sensitivity between all node-pairs, and not just those separated by L-hops:

$$\left\| \frac{\partial \mathbf{z}_i^{(L)}}{\partial \mathbf{x}_j} \right\|_1 \leq \zeta_2^L \left(\sum_{l=0}^{L} \hat{\mathbf{A}}^l \right)_{ij}, \ \forall \, (i, j) \in \mathcal{V}$$

With `DropEdge`, the bound changes to:

$$\mathbb{E}_{\mathbf{M}^{(1)},\ldots,\mathbf{M}^{(L)}} \left[\left\| \frac{\partial \mathbf{z}_i^{(L)}}{\partial \mathbf{x}_j} \right\|_1 \right] \leq \zeta_2^L \left(\mathbb{E}_{\mathbf{M}^{(1)},\ldots,\mathbf{M}^{(L)}} \left[\sum_{l=0}^{L} \hat{\mathbf{A}}^l \right] \right)_{ij} = \zeta_2^L \left(\sum_{l=0}^{L} \left(\mathbb{E}_{\mathbf{M}} \left[\hat{\mathbf{A}} \right] \right)^l \right)_{ij}$$

Figure 3.7 shows the plot of the entries of $\sum_{l=0}^{6} \dot{\mathbf{P}}$, as in the upper bound above with $\hat{\mathbf{A}} = \hat{\mathbf{A}}^{\text{asym}}$, against the shortest distance between corresponding nodes. We observe that the sensitivity between nearby nodes marginally increases, while that between distant nodes significantly decreases. Similar observations are made with $\hat{\mathbf{A}} = \hat{\mathbf{A}}^{\text{sym}}$.

Finally, Theorem 2.6 considers the case of `ReLU-GCNs` with $\hat{\mathbf{A}} = \hat{\mathbf{A}}^{\text{asym}}$ for message passing. Moreover, it assumes that each path in the computational graph is active with a fixed probability. In this case, the sensitivity (in expectation) between any pair of nodes is given by

$$\left\|\mathbb{E}\left[\frac{\partial \mathbf{z}_i^{(L)}}{\partial \mathbf{x}_j}\right]\right\|_1 = \left(\hat{\mathbf{A}}^L\right)_{ij}\left[\rho\left\|\prod_{l=1}^{L}\mathbf{W}^{(l)}\right\|_1\right] = \zeta_3^{(L)}\left(\hat{\mathbf{A}}^L\right)_{ij}$$

Following the steps in Equation 3.11,

$$\mathbb{E}_{\mathbf{M}^{(1)},\ldots,\mathbf{M}^{(L)}}\left[\left\|\mathbb{E}\left[\frac{\partial \mathbf{z}_i^{(L)}}{\partial \mathbf{x}_j}\right]\right\|_1\right] = \zeta_3^{(L)}\left(\left(\mathbb{E}_{\mathbf{M}}\left[\hat{\mathbf{A}}\right]\right)^L\right)_{ij}$$

Here, again, we invoke Theorem 3.3 to conclude that using `DropEdge` reduces the expected sensitivity of a node's representations to the features of nodes L-hops away. Empirical observations in Figure 3.5 and Figure 3.6 suggest that we should expect an increase in sensitivity to neighboring nodes, but a decrease in sensitivity to those farther away.

3.2.4 Monte-Carlo DropEdge

In the previous sections, we focused on the expected sensitivity of the stochastic representations:

$$\mathbb{E}_{\mathbf{M}}\left[\left\|\frac{\partial \mathbf{z}_i^{(L)}}{\partial \mathbf{x}_j}\right\|_1\right]$$

This corresponds to the training-time behavior of `DropEdge`, wherein model activations are random. Another way to study `DropEdge`'s effect is by examining the sensitivity of the *expected representations*:

$$\left\|\frac{\partial}{\partial \mathbf{x}_j}\mathbb{E}_{\mathbf{M}}\left[\mathbf{z}_i^{(L)}\right]\right\|_1$$

This approach corresponds to test-time Monte-Carlo averaging over the `DropEdge` ensemble (MC-DE), which is better at alleviating over-smoothing as well as at generalization [XZL23]. With a linear model, the order of the two operations – expectation w.r.t. `DropEdge` and sensitivity computation – was irrelevant:

$$\mathbb{E}_{\mathbf{M}}\left[\left\|\frac{\partial \mathbf{z}_i^{(L)}}{\partial \mathbf{x}_j}\right\|_1\right] = \left\|\mathbb{E}_{\mathbf{M}}\left[\bar{\mathbf{A}}_{ij}\right]\bar{\mathbf{W}}\right\|_1 = \left\|\mathbb{E}_{\mathbf{M}}\left[\frac{\partial \mathbf{z}_i^{(L)}}{\partial \mathbf{x}_j}\right]\right\|_1 = \left\|\frac{\partial}{\partial \mathbf{x}_j}\mathbb{E}_{\mathbf{M}}\left[\mathbf{z}_i^{(L)}\right]\right\|_1$$

In general, however, the two quantities can be related using the convexity of norms and Jensen's inequality:

$$\left\| \frac{\partial}{\partial \mathbf{x}_j} \mathbb{E}_{\mathbf{M}} \left[\mathbf{z}_i^{(L)} \right] \right\|_1 \leq \mathbb{E}_{\mathbf{M}} \left[\left\| \frac{\partial \mathbf{z}_i^{(L)}}{\partial \mathbf{x}_j} \right\|_1 \right]$$

Therefore, the discussion in Section 3.2.3 extends to the MC-DE representations as well. Although tighter bounds can possibly be derived for this setting, we will leave that for future works.

Chapter 4

Experiments

Our theoretical results in Chapter 3 suggest that DropEdge must worsen the performance of GNNs on tasks relying on the model's capacity to capture LRIs. In this chapter, we will conduct a set of experiments to validate this hypothesis. In Section 4.1, we propose a measure of the average distance to which signal from a source node is propagated. In a training-free setting, we show that the propagation distance is negatively correlated with total commute time in a graph, and using DropEdge reduces it further. In Section 4.2, we reproduce the SyntheticZINC experiment from [Gio+24]. Furthermore, we show that DropEdge leads to a performance degradation at all levels of node-mixing. Finally, in Section 4.3, we show that the performance of GNNs, as the DropEdge probability increases, exhibits starkly contrasting trends between citation networks (Cora [McC+00] and CiteSeer [GBL98]) and molecular datasets (Proteins [DD03] and MUTAG [KMB05]). Accordingly, we propose a novel hypothesis on the effect of DropEdge: on short-range tasks, DropEdge improves the performance of GNNs by reducing their receptive field, thereby increasing model-dataset alignment. On long-range tasks, on the other hand, it reduces this alignment, negatively impacting model performance.

4.1 Signal Propagation

In this section, we construct a synthetic experiment with real-world molecular datasets, `Proteins` and `MUTAG`, to validate the theoretical results characterizing the relationship between commute times in the graph and the extent of signal propagation in an `MPNN`. We closely follow the setup laid out in [Di +23, Appendix F], with a few changes. Particularly, for a given graph $\mathsf{G}(\mathcal{V}, \mathcal{E})$, we start by sampling a source node $v \in \mathcal{V}$ and setting its features to a $H^{(0)}$-dimensional unitary vector (whose entries sum to 1), where $H^{(0)}$ is the dimension of the original input features. The features of all other nodes are set to $\mathbf{0}_{H^{(0)}}$. We use L-layer `GCNs` to compute a measure of signal propagation in the graph, and show that it negatively correlates with the expected commute time.

4.1.1 Propagation Distance

The measure of signal propagation proposed in [Di +23] was

$$\tilde{z}_{\odot}^{(L)} := \frac{1}{H^{(L)}} \sum_{d=1}^{H^{(L)}} \sum_{i=1}^{N} \left[\left(\frac{\mathbf{z}_{i,d}^{(L)}}{\left\| \mathbf{z}_{i}^{(L)} \right\|} \right) \left(\frac{d_{\mathsf{G}}(i,j)}{\max_{u \in \mathcal{V}} d_{\mathsf{G}}(u,v)} \right) \right] \tag{4.1}$$

The term $\mathbf{z}_{i,d}^{(L)}$ provides a measure of the signal received at node i along dimension d. The feature values are normalized to offset the possible inflation of the unit mass by the randomly initialized model, as it is propagated across the graph. These values are used to weigh the distance $d_{\mathsf{G}}(i,j)$ from the source node, which are also normalized so as to reduce the effect of the graph topology – in the absence of normalization, this measure would be misleadingly low for graphs with small pairwise distances. Then, for each dimension $d \in \left[H^{(L)}\right]$, the inner summation corresponds to the average distance the signal has propagated to. Finally, $\tilde{z}_{\odot}^{(L)}$ is computed as the mean of the propagation distances over all dimensions. We make a few notes about this measure:

- The measure of the signal received at a given node is *dependent* on the *sign* of the feature values. This is an incorrect choice since it implies negative feature values have lesser information than positive ones.

- The normalization of the feature matrix is performed over the nodes. We argue that this loses the disparity in the amount of signal propagated to different nodes. For example, assume that a GNN outputs the feature vectors $\mathbf{z}_i^{(L)} = a\mathbf{1}_{H^{(L)}}$ and $\mathbf{z}_j^{(L)} = b\mathbf{1}_{H^{(L)}}$, with $a, b \in \mathbb{R}$ and $|a| > |b|$, for some nodes $i, j \in \mathcal{V}$. Normalizing them over the nodes would reduce them to the same vector, $\left(1/H^{(L)}\right)\mathbf{1}_{H^{(L)}}$, failing to account for the fact that $\mathbf{z}_i^{(L)}$ is more signal-rich than $\mathbf{z}_j^{(L)}$.

- The distances are normalized by the maximum distance from the source node. In the case of large graphs where $\max_{u\in\mathcal{V}} d_{\mathsf{G}}(u, v) > L$, the signal cannot even reach the nodes beyond L-hops from the source yet the normalization term takes these nodes into account. This erroneously biases the propagation distance to be lower for larger graphs.

To correct these shortcoming, we define our measure of signal propagation through the L-layer GNN as

$$\tilde{z}_{\odot}^{(L)} := \frac{1}{H^{(L)}} \sum_{d=1}^{H^{(L)}} \sum_{i=1}^{N} \left[\left(\frac{\left|\mathbf{z}_{i,d}^{(L)}\right|}{\sum_{j\in[N]} \left|\mathbf{z}_{j,d}^{(L)}\right|} \right) \left(\frac{d_{\mathsf{G}}(i, v)}{\min\left(L, \max_{j\in[N]} d_{\mathsf{G}}(j, v)\right)} \right) \right] \tag{4.2}$$

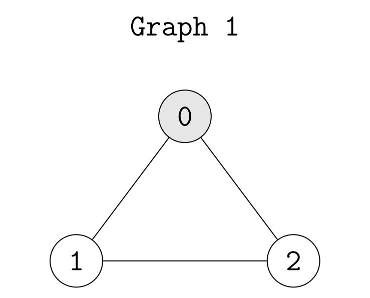
Our proposed measure makes the following changes to the one proposed by [Di +23] (Equation 4.1):

- We use the absolute values of the features to weigh the shortest distances, instead of the raw values, since these are more appropriate for measuring the signal contained in them.

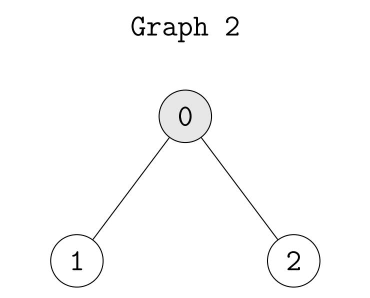
- The normalization of the feature matrix is performed over the nodes, instead of over the feature dimensions. This would retain the ordering of the nodes along each dimension, while simultaneously ensuring that the average path lengths are not artificially inflated by the (random) weights of the network.

- The distances are normalized by the maximum distance from the source that the signal *can* reach, so that the propagation distance is not misleadingly low for large graphs.

To provide empirical support for our measure, we consider two different sets of graphs and compute the following quantities: 1. sum of effective resistances between the source node and all other nodes, 2. total commute time from the source node, 3. the propagation distance in Equation 4.1, which we call the *original measure*, and 4. the propagation distance with our proposed changes, as in Equation 4.2, which we refer to as the *proposed measure*. To keep things simple, we initialize the features of the source node to $\mathbf{x}_v = 0.5 \times \mathbf{1}_2 \in \mathbb{R}^2$, so that it is a deterministic unitary vector. We use a 2-layer `GCN` with both weight matrices equal to identity, $\mathbf{W}^{(1)} = \mathbf{W}^{(2)} = \mathbf{I}_2$. Furthermore, we only use graphs in which the source is at most 2-hops away from all other nodes, so that under-reaching [Bar+20] is not an issue.



Effective Resistance $= 1.33$
Commute time $= 8.00$
Original measure $= 1.00$
Proposed measure $= 0.67$



Effective Resistance $= 2.00$
Commute time $= 8.00$
Original measure $= 1.00$
Proposed measure $= 0.60$

In the set of graphs above, the gray-colored nodes are the source nodes, with their features set to $\mathbf{x}_v$. Let's consider graph 1. In the first `MPNN` layer, a signal is sent from node 0 to node 1 and node 2. In the second `MPNN` layer, signals from node 0 and node 1 are sent to node 2, and signals from nodes node 0 and node 2 are sent to node node 1. However, in graph 2, the second layer does not allow communication of the (non-zero) signal between node 1 and node 2. As a result, we would expect the propagation distance to be shorter in the second graph. This expectation is supported by the fact that the total resistance and total commute time (from the source) are higher in graph 2. While the original measure is the same in both graphs, our measure captures the effect of the topological changes and decreases accordingly.

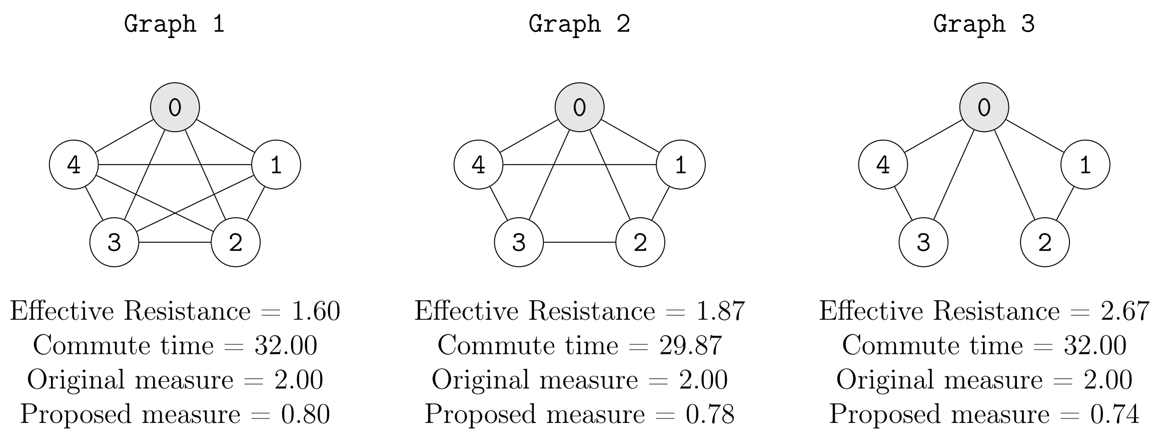


Figure 4.1: A sequence of 5-node connected graphs.

We will look at another set of graphs obtained by successively removing a pair of edges from the 5-complete graph (left-most), while ensuring that the source stays connected to all other nodes. As we saw with the previous set, the original measure cannot discriminate between these graphs even though the total resistance increases. However, our measure is able to do so, demonstrating its better discriminative capacity.

Observe that while the total resistance increases over both the edge removal steps, as expected (Theorem 2.2), the total commute times do not have a similar monotonic relation. This makes our effort to demonstrate a negative correlation between total commute time and propagation distance non-trivial, i.e. it is not immediately obvious that the same observations, as in [Di +23], will be made.

4.1.2 Effect of DropEdge

Propagation distance describes the average distance a unit mass has travelled from its starting point; a higher propagation distance suggests that, on average, the unit mass has journeyed further from the source node. While [Di +23] shows its relation with effective resistance, we extend the analysis to commute times. Specifically, for a given dataset $\mathcal{D}$,

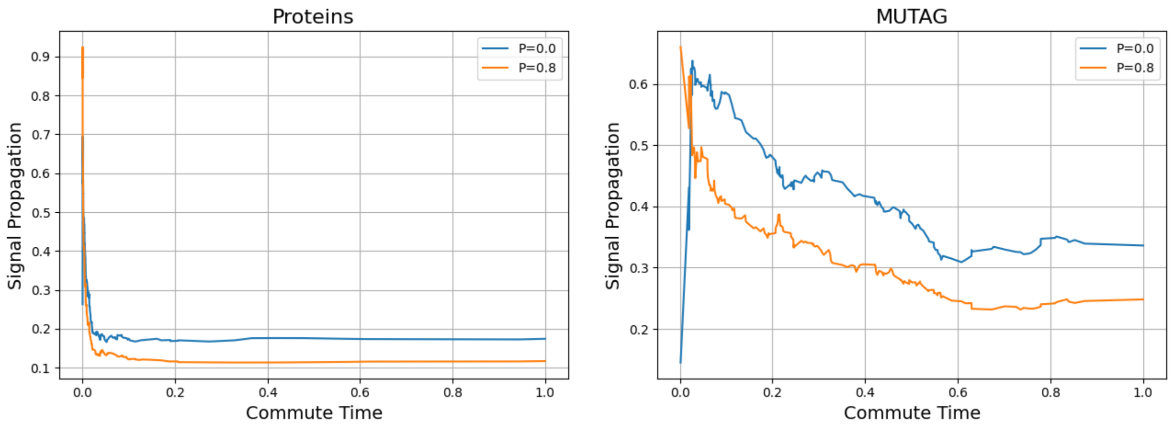


Figure 4.2: Propagation distance (our measure, Equation 4.2) versus total commute times in molecular datasets.

we sample up to 200 graphs[1], $\mathsf{G}^i\left(\mathcal{V}^i, \mathcal{E}^i\right) \in \mathcal{D}$, and perform 10 runs with each of them. In each run, a source node, $v \in \mathcal{V}^i$ is sampled at random and its features are set to a random unitary vector, i.e. $\left\|\mathbf{x}_v\right\|_1 = 1$, thus creating a *unit mass*. We use randomly initialized `GCNs` with the depth set at $L = 10$, which is close to the average diameter of the graphs in the datasets. The width of all the hidden layers is fixed at $H^{(1)} = \ldots = H^{(L)} = 5$, and `ReLU` activation is used. The intercept term is removed from all the message-passing layers so that the node representations in the `GCN`'s output are non-zero only if they have received information from the source node. For a `DropEdge` model, 10 stochastic forward passes are averaged to compute the feature vectors, $\mathbf{Z}^{(L)}$. The propagation distance, $\tilde{z}_{\odot}^{(L)}$, is computed using Equation 4.2, and the total commute time from the source node, defined as the sum of commute times between the source and all nodes *at most* L-hops away, is computed using Theorem 2.3. The propagated distances and total commute times are averaged over the 10 runs, each with a different source node, to create a datum corresponding to G^i.

Figure 4.2 shows the plots of propagation distances versus total commute times for a `NoDrop` model and a `DropEdge` model with $p = 0.8$. The first thing to note is that the propagation distance is lower in graphs with higher commute times, in agreement with the results in [Di +23]. Secondly, employing `DropEdge` *decreases the propagation capacity* of the `GCN`, thereby providing empirical evidence for our theoretic results in Section 3.1. We do emphasize that these results are for random networks, and do not account for the effects of `DropEdge` on `GNN` training; we will address this case in Section 4.2.

[1]MUTAG has only 188 graphs, so we use all of them.

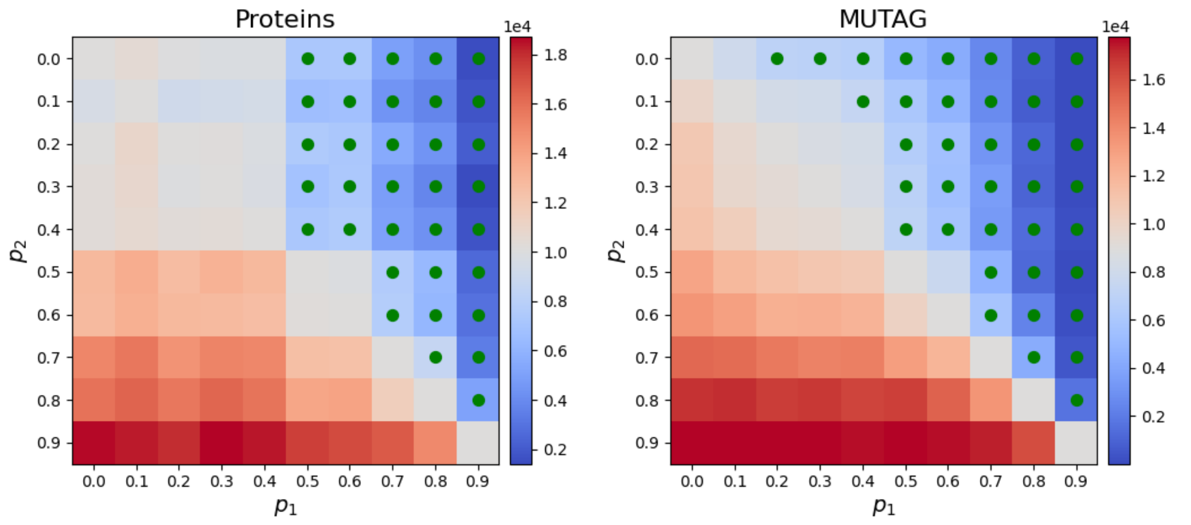


Figure 4.3: Wilcoxon signed-rank test-statistics between propagation distances (our measure, Equation 4.2) for `DropEdge` models with $p = 0.0, \ldots, 0.9$. Green dots indicate significant results at 95% confidence level.

To provide further evidence for this effect of `DropEdge`, we perform the pairwise Wilcoxon signed-rank test between the propagation distances for models using `DropEdge` probabilities $p = 0.0, 0.1, \ldots, 0.9$. We set the alternate hypothesis to be that the first sample is, element-wise, *less* than the second sample, i.e. a low test-statistic (more significant) suggests that the $\tilde{z}_{\odot}^{(L)}$ samples from a `DropEdge` model with $p = p_1$ are consistently lower than the paired ones from a model with $p = p_2$. The heat maps of the pairwise test-statistics are shown in Figure 4.3. Most of the tests with $p_1 > p_2$ record significant evidence for rejecting the null hypothesis in the favor of the alternate, providing further evidence towards a higher `DropEdge` probability reducing the propagation distance in a `GCN`.

For the sake of completeness, we also add the plots with the original measure of propagation distance (Equation 4.1), reproducing the experiment in [Di +23]. The results are shown in Figure 4.4. We observe that the trends are completely different from the ones in Figure 4.2, as well as those reported in [Di +23, Figure 5]. This is because the implementation of their experiment has a small error[2] – the feature normalization is performed using a raw sum of all the entries in the feature matrix, i.e. $\sum_{i=1}^{N} \sum_{d=1}^{H^{(L)}} \mathbf{z}_{i,d}^{(L)}$, instead of a node-wise normalization using $\left\| \mathbf{z}_i^{(L)} \right\|_1$.

[2]https://github.com/lrnzgiusti/on-[]oversquashing/blob/322fc4c6371a29dc48c1b54232b914006a1db481/exp/signal_propagation.py#L67

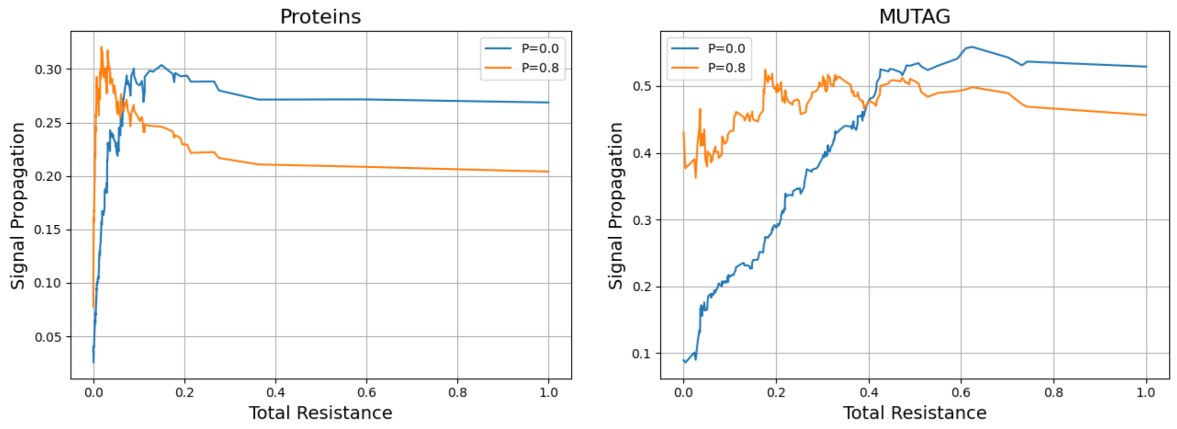


Figure 4.4: Propagation distance (original measure, Equation 4.1) versus total resistance in molecular datasets.

4.2 SyntheticZINC

In Section 4.1, we showed that `DropEdge` reduces the propagation distance in an `MPNN`, which suggests that it has a detrimental effect on `MPNNs` learning LRIs. However, our experiment did not account for the effect of `DropEdge` on model training. Although it is hard to precisely characterize the effect of `DropEdge` on the training dynamics of an `MPNN`, we can study it by comparing the performance of a regular `MPNN` against one using `DropEdge`, keeping all else equal. In this experiment, we 1. reproduce the results from [Gio+24], demonstrating that `MPNNs` perform worse as the minimum distance between nodes with non-zero mixing in the ground-truth increases, and 2. show that using `DropEdge` hurts the performance of an `MPNN` by reducing the level of mixing it can express.

4.2.1 Experimental Setup

Since it is difficult to determine the level of node-mixing in real-world datasets, following [Gio+24, Section 5], we generate synthetic node-level features and graph-level labels, controlling the mixing in the ground-truth. Specifically, given a graph $\mathsf{G}(\mathcal{V},\mathcal{E})$, we set all the node features to 0, except for two nodes', i and $j \neq i$, whose features are sampled as $x_i, x_j \in \mathcal{U}(0,1)$. The graph-level target is computed as $y = \tanh(x_i + x_j)$, i.e. the task requires a non-linear mixing between the features of nodes i and j. These nodes are chosen to induce the desired level of underlying mixing – given $\alpha \in [0, 1]$, the node-pair (i, j) is chosen such that $\mathbf{C}_{ij}$ is the α-quantile of the distribution of commute times over G. The graph samples are taken from the `ZINC` chemical dataset [Irw+12], with the dataset size constrained to 12K molecular graphs [Dwi+23]. This way, our dataset is only *semi-synthetic*, making the results of this experiment more reliable. We refer to this dataset as `SyntheticZINC`.

We analyze the effect of underlying mixing on model performance by varying α between 0 and 1. The `MPNN` is chosen to be an L-layer `GCN` with a `MAX`-pooling readout – this encourages the model to learn the mixing by passing messages effectively [Gio+24, Theorem 3.2]. The model depth is set at $L = \max_{\mathsf{G}} \lceil \operatorname{diam}(\mathsf{G})/2 \rceil = 11$ to ensure that the `GCN` does not suffer from under-reaching [Bar+20; AY21]. We experiment with a baseline `NoDrop` model, and a `DropEdge` model with $P = 0.5$. The models are trained using the Adam optimizer [KB15], with a learning rate of 2×10^{-3} and a weight decay of 1×10^{-4}, for a total of 250 epochs. We perform 10 runs with each configuration to account for the randomness in training, e.g. in initialization and mini-batch sampling. When reporting the training metrics, we present the best performance achieved during the 250 epochs of training. The test metrics are calculated using early stopping, based on the best validation set metrics.

4.2.2 Results

[Gio+24, Theorem 4.4] suggests that over-squashing in `MPNNs` depends heavily on the commute time of the underlying mixing, and the results in Section 3.1 suggest that `DropEdge` uniformly increases the commute times in the computational graph. Therefore, we expect to make two observations: 1. the metrics worsen as α is increased from 0 to 1, and 2. the `DropEdge` model performs worse than the `NoDrop` model.

The Mean Absolute Error (MAE) over the training and testing sets are shown in Figure 4.5. As expected, the performance of the `NoDrop` model worsens as α increases, i.e. as the underlying function entails mixing between farther off nodes. By fixing the `GNN` architecture, we controlled for the effects of vanishing gradients and over-smoothing, so that the cause of declining performance can be fully attributed to over-squashing.

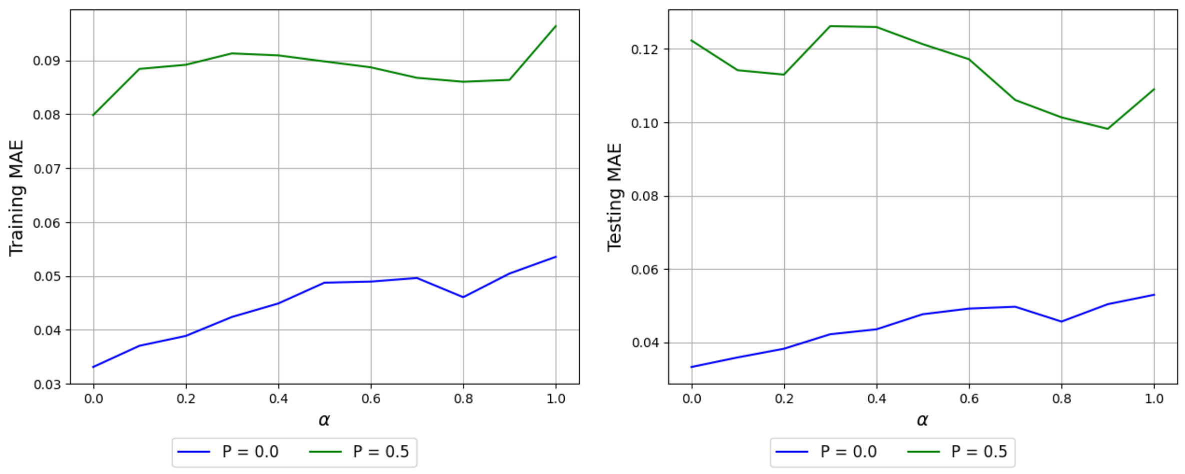


Figure 4.5: Train and Test MAE of `GCN` on the `SyntheticZINC` dataset, with varying levels of the underlying mixing

The performance of the `DropEdge` model is much worse than the `NoDrop` baseline, implying that `DropEdge` significantly reduces the mixing between nodes. Finally, note that the `NoDrop` model has a near-zero generalization gap (difference between the training and testing MAE), while it is *much higher* for the `DropEdge` model. This is an unintuitive result since `DropEdge` famously reduces over-fitting on short-range tasks [Ron+20]. While the reason for this observation is unclear, [AY21] hypothesizes that this is because the model overfits to short-range artifacts in the training data, leading to poor generalization. This observation suggests a need for a re-evaluation of the suitability of `DropEdge` (and other similar methods designed to train deep `GNNs`) at learning long-range tasks.

4.3 Real-World Datasets

In Section 3.2, we concluded that `DropEdge` increases sensitivity to nearby nodes, while reducing it between nodes far from each other. This would imply that `GNNs` trained on homophilic datasets – e.g. citation networks like `Cora` and `CiteSeer` – would benefit from `DropEdge`, especially if these `GNNs` are deep. On the other hand, `DropEdge` could be expected to worsen `GNNs` on long-range tasks – e.g. molecular datasets like `Proteins` and `MUTAG`. To validate these hypotheses, we study the effect of `DropEdge` probability on training accuracy of the `GCN` and `GAT` models.

Name	**#graphs**	**#nodes**	**#edges**	**#features**	**#classes**
Cora	1	2,708	10,556	1,433	7
CiteSeer	1	3,327	9,104	3,703	6
Proteins	1,113	∼39.1	∼145.6	3	2
MUTAG	188	∼17.9	∼39.6	7	2

Table 4.1: Dataset Statistics. Note that the number of edges is counts each direction of an undirected edge twice (for fair comparison between directed and undirected graphs).

Description of the Datasets: `Cora` and `CiteSeer` are citation networks – their nodes represent scientific publications and an edge between two nodes indicates that one of them has cited the other. The features of each publication is represented by a binary vector, where each index indicates whether a specific word from a dictionary is present or absent. The `Proteins` dataset is a molecular property prediction dataset where the task entails classifying protein molecules as enzymes or not. Similarly, the `MUTAG` dataset consists of nitroaromatic compounds, and the task is to predict their mutagenic effects on *Salmonella typhimurium*. The statistics of these four datasets are presented in Table 4.1.

4.3.1 Experimental Setup

We train randomly initialized L-layer `GCNs` and `GATs` with $K = 2$ attention heads, on `Cora`, `CiteSeer`, `Proteins` and `MUTAG` datasets. The number of message passing steps is varied from $L = 2$ to $L = 7$ for `Cora` and `CiteSeer`, but only till $L = 6$ for `Proteins` and `MUTAG` since deeper `GNNs` struggled to learn on these datasets. The size of the hidden representations is set at 64 for the citation networks and 32 for the molecular datasets. A linear readout layer is used to compute the node-level logits for citation networks, whereas for the molecular datasets, the readout layer is a composition of the `AVG`-pooling layer and a linear output layer. The `DropEdge` probability is varied from $q = 0.1$ to $q = 0.9$, in increments of 0.1. The models are trained using the Adam optimizer, with a learning rate of 5×10^{-3} and a weight decay of 5×10^{-4}, for a total of 300 epochs. We conduct 5 runs with each hyperparameter setting and then average the metrics.

4.3.2 Citation Networks

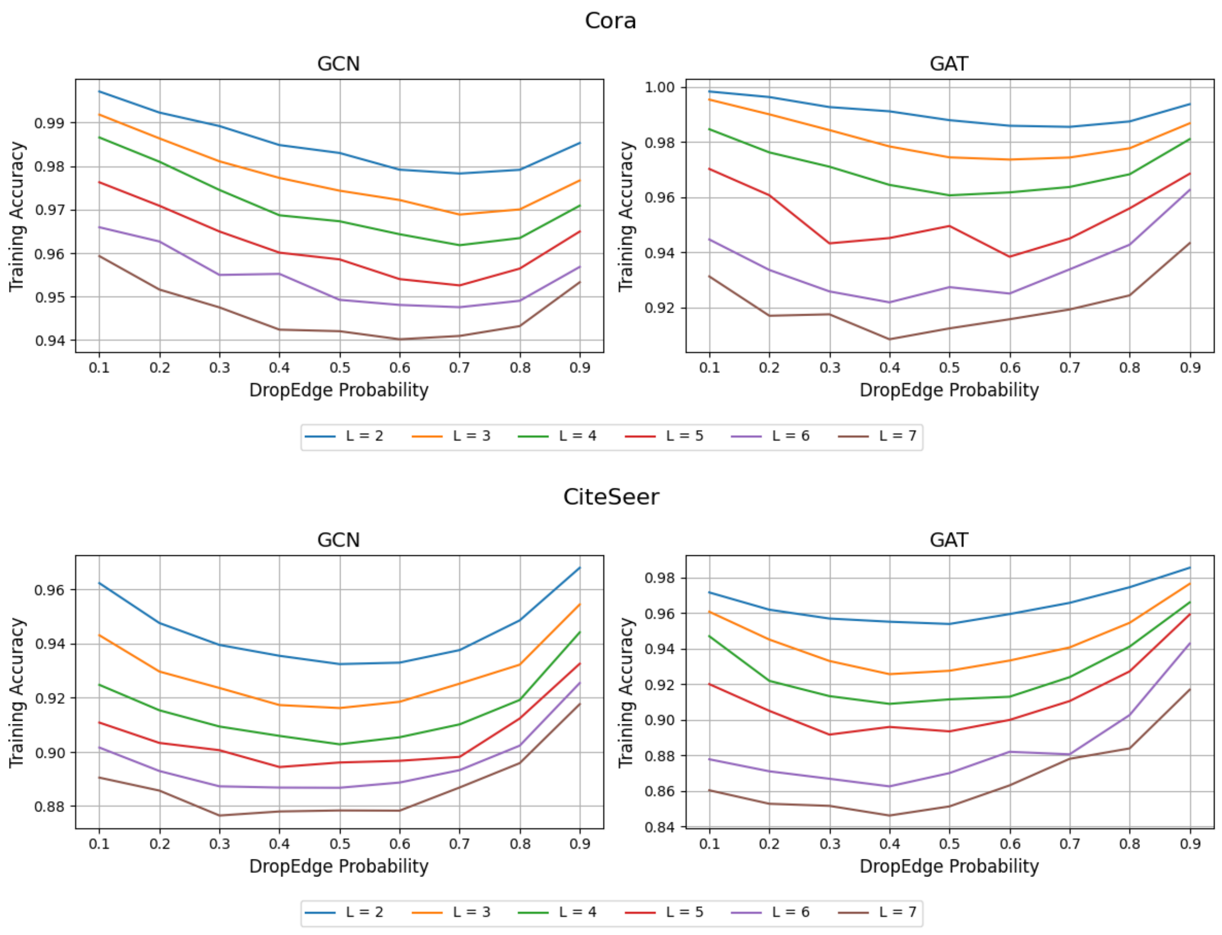


Figure 4.6: Training accuracy versus the `DropEdge` probability for the citation networks.

The training accuracy versus `DropEdge` probability plots for `Cora` and `CiteSeer` datasets are shown in Figure 4.6. The trends in both the plots convey an interesting story. While we expected the models to benefit from increasing `DropEdge` probability q, in fact, the training accuracy falls up to a point, before increasing. This can be explained as follows – as the `DropEdge` probability is increased from $q = 0$, although the model gets aligned with the short-range tasks by reducing the effective receptive field of the `MPNNs`, the underlying tasks also become harder as the entropy of the input distribution increases. That is, there is an interplay of two opposing effects – on one hand, the increased alignment should improve model performance, but on the other, the increased difficulty of the learning task should worsen it. Once q becomes large enough, the entropy of the input distribution starts reducing again and the complexity of the training task starts reducing. Accordingly, the

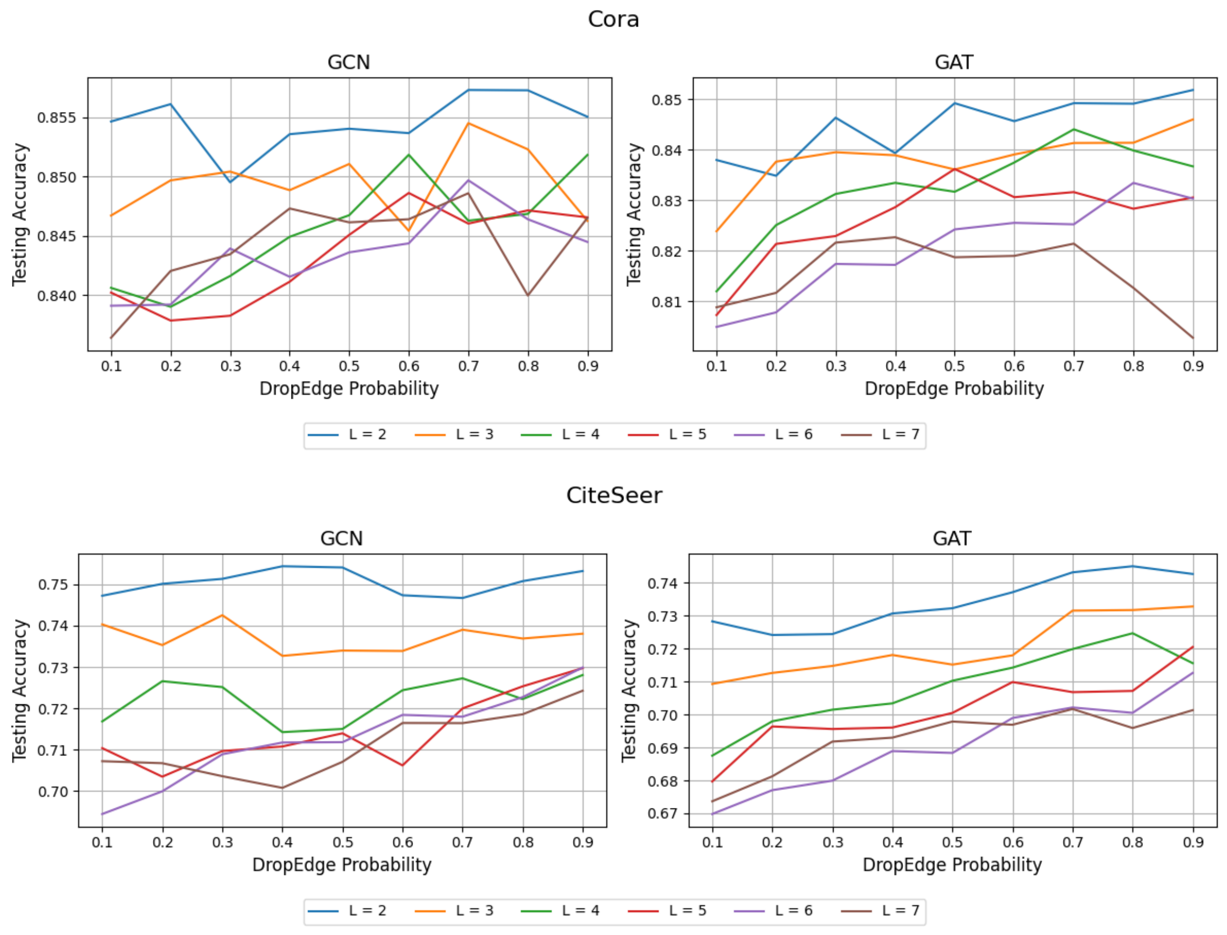


Figure 4.7: Testing accuracy versus the `DropEdge` probability for the citation networks.

two effects start complementing each other and the performance improves. Another observation we make is that the minima of training accuracy is achieved at a lower `DropEdge` probability for deeper networks, than for shallower ones. This implies that the benefits from model alignment start to dominate sooner for deeper models, and sometimes their performance improves even before the entropy of the data starts decreasing. This supports our hypothesis that the benefits of `DropEdge` are more significant when it comes to aligning deep `MPNNs` for short-range tasks, but less so at aligning shallow `MPNNs`.

As for the test accuracy (shown in Figure 4.7), we note that it increases as the `DropEdge` probability is increased. This can, again, be explained as two complementary effects. Firstly, the reduced receptive field during training time biases the convergence towards locally consistent models, thereby aligning the learnt model with the ground-truth and reducing the approximation error. Second is the reduction in over-fitting thanks to a stochastically augmented dataset, similar to how random image augmentation techniques, like rotation, cropping and/or flipping, improve generalization in CNNs [SK19].

A final observation we make is on the effect of model depth. Clearly, both training and testing metrics decrease as the model depth is increased. This can be explained by several complementary factors:

- Training is more unstable for deeper `MPNNs`, with vanishing gradients and over-smoothing posing a major constraint towards training deeper models. While `DropEdge` reduces the extent of over-smoothing, it still does not alleviate it.
- Moreover, we hypothesize that the higher receptive field of deep `MPNNs` decreases their alignment with the citation networks, thereby negatively affecting performance.

Though our novel hypothesis (in the last point) cannot be clearly validated in the presence of the other confounding effects, we'll make a reliable conclusion using these results in combination with the ones for long-range molecular datasets.

4.3.3 Molecular Datasets

The training accuracies with the molecular datasets, `Proteins` and `MUTAG`, are shown in Figure 4.8. The trends here are a lot different from those observed for citation networks. Specifically, the training accuracy has a decreasing trend against the `DropEdge` probability. We explain this by adapting the hypothesis we laid out for the citation networks: for long-range tasks, `DropEdge` contributes in two detrimental ways – one is by reducing the effective receptive field of the `MPNN`, thereby *reducing the model alignment*, and the other is by making the learning task harder.

Similarly, the test accuracy, as shown in Figure 4.9, exhibits a decreasing trend, which contrasts sharply with the behavior observed for citation networks in Figure 4.7. Furthermore, the impact of increasing model depth on molecular datasets appears ambiguous, despite the well-known issues of vanishing gradients and over-smoothing. This ambiguity

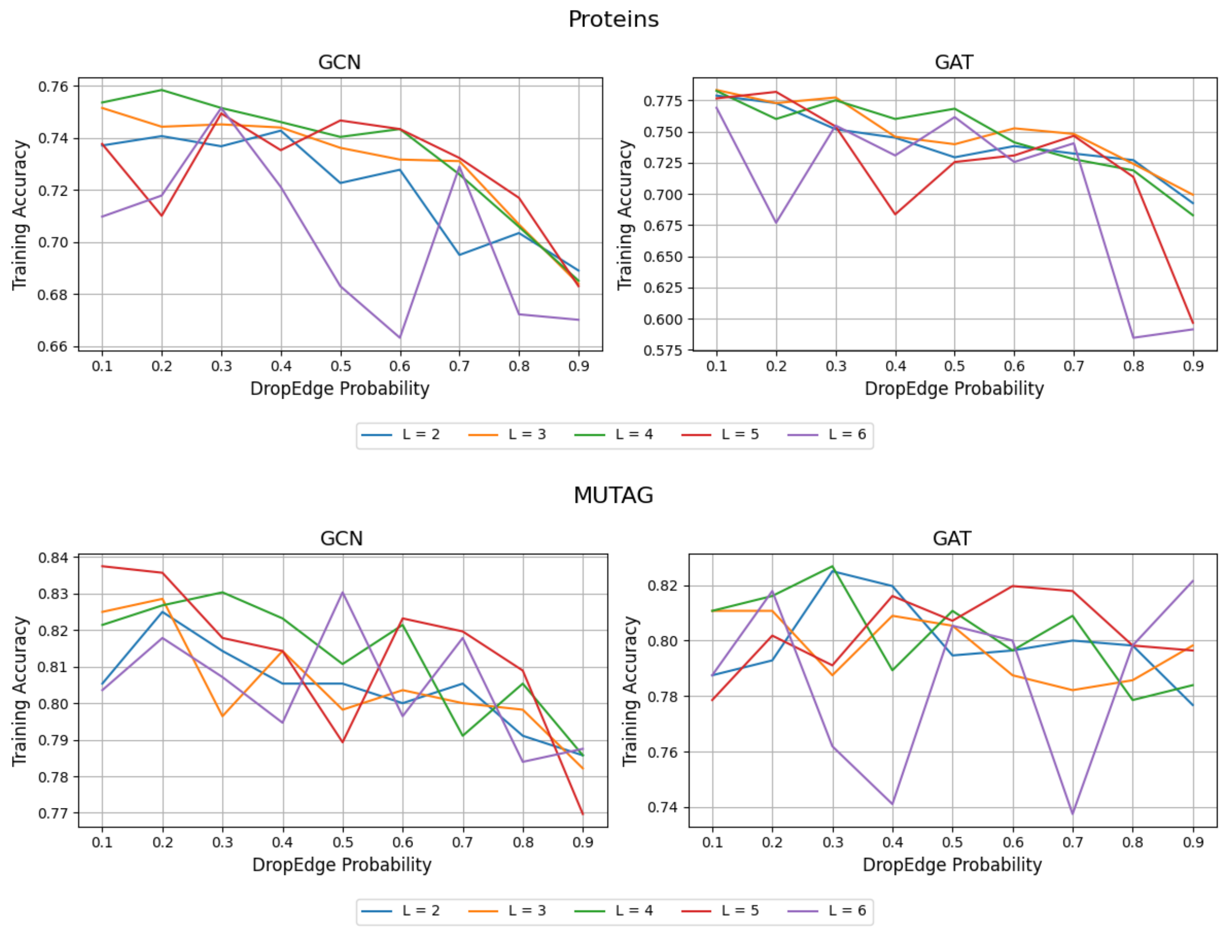


Figure 4.8: Training accuracy versus the `DropEdge` probability for the molecular datasets.

arises because, for molecular datasets, *deeper models better align with the task*, partially mitigating these detrimental effects and enabling deeper `MPNNs` to remain competitive with shallower ones. These trends do not align with the existing explanations of `DropEdge`'s effects on over-smoothing and over-fitting; however, our hypothesis offers a more fitting explanation.

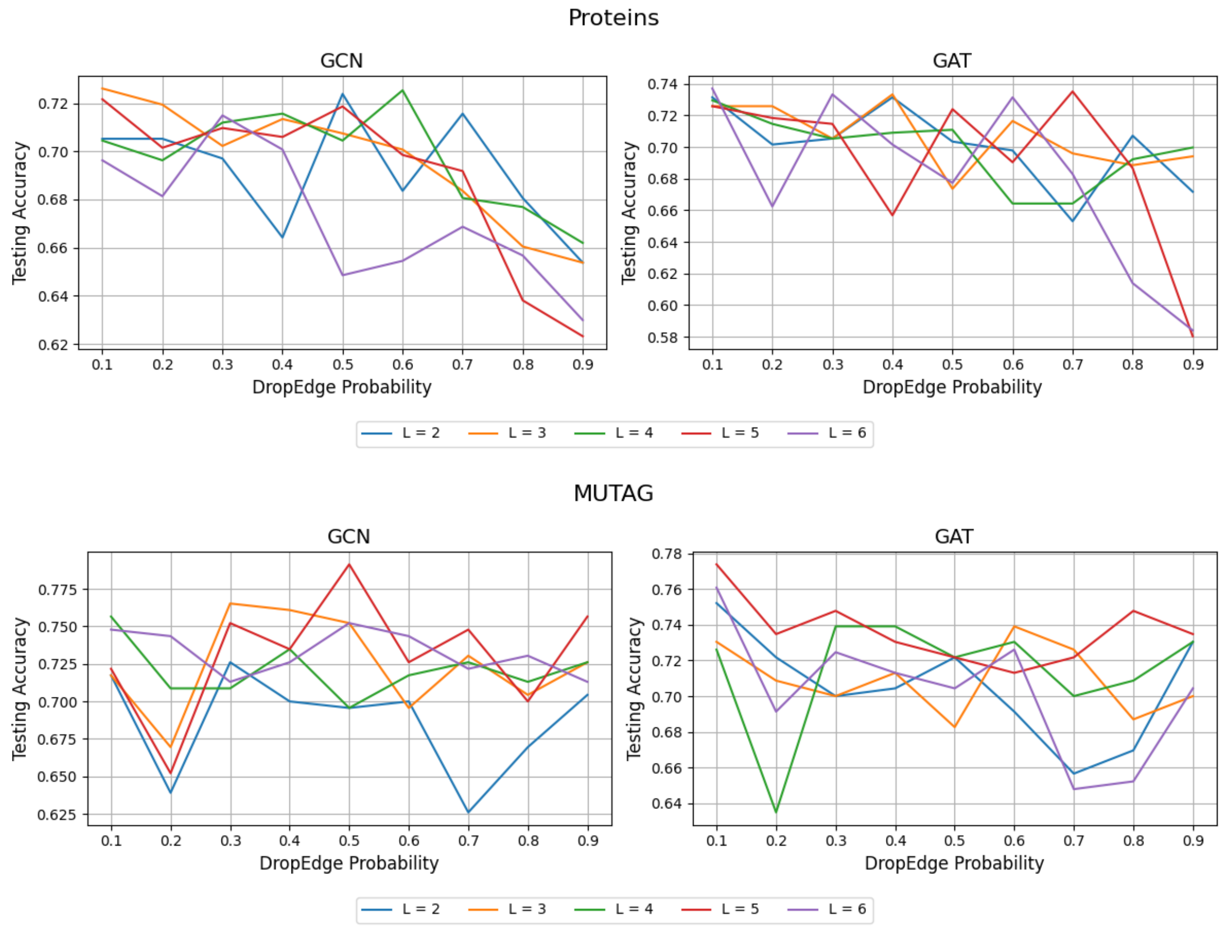


Figure 4.9: Testing accuracy versus the `DropEdge` probability for the molecular datasets.

Chapter 5

Conclusion

The objective of our work was to explore the impact of `DropEdge` on over-squashing in `MPNNs`. Specifically, we aimed to determine how `DropEdge` influences model performance on tasks that entail long-range information propagation. To achieve this, we approached the problem from two complementary perspectives: 1. we studied the effects of `DropEdge` on the expected commute time, which is indicative of the mixing capacity of an `MPNN`, and 2. we conducted an analysis of the effect of `DropEdge` on the sensitivity of node representations and evaluated its impact on performance metrics on both synthetic and real-world datasets. Through these investigations, we aimed to better understand how `DropEdge` might effect the over-squashing levels, as well as influence the performance of `GNNs` on long-range learning tasks.

5.1 Key Findings

In this work, we studied the effect of `DropEdge` on over-squashing in `GNNs`, and investigated its applicability at modelling LRIs. In Section 3.1, we showed that `DropEdge` increases the expected commute time between all node-pairs, suggesting that it reduces the mixing of information between distant nodes. In Section 3.2, we studied the sensitivity of node representations in an L-layer `GCN`, showing that it is reduced between nodes L-hops away.

This implies that DropEdge reduces the model's effective receptive field. Furthermore, we extended the previously established sensitivity bounds for nonlinear MPNNs to the DropEdge setting, making similar conclusions as in the case of linear models. In Section 4.1, we presented empirical support for our theoretical findings by showing that DropEdge reduces the signal propagated by a GCN. Towards a more performance-based evaluation of DropEdge, in Section 4.2, we showed that it significantly increases the MAE on the training set, as well as that on the test set, in a (semi-synthetic) long-range task. Moreover, the generalization gap of the DropEdge model was observed to be significantly larger, which is in stark contrast to its reported effect on short-range tasks [Ron+20]. Finally, in Section 4.3.2, we evaluated the effect of DropEdge on real-world (short-range) citation networks [GBL98; McC+00], and in Section 4.3.3, on (long-range) molecular datasets [DD03; KMB05]. We proposed a novel hypothesis justifying the noticeably different trends in the two cases – while DropEdge improves the test metrics on short-range tasks by reducing the GNN's receptive field, thereby increasing model-dataset alignment, it degrades the generalizability of the models on long-range tasks by forcing them to overfit to short-range artifacts from the training set.

Our analysis points out a key assumption in algorithms designed for training deep GNNs: the idea that if a deep GNN is trainable, it must also have the ability to model LRIs. In other words, that alleviating the problems of vanishing gradients and over-smoothing is sufficient to ensure that deep GNNs can be effective at long-range tasks. Our results suggest that this, in fact, need not be true – we theoretically and empirically show that DropEdge exacerbates the over-squashing problem in deep GNNs, and degrades their performance on long-range tasks. Our results highlight a need for a thorough evaluation of DropEdge, and other methods employed when training deep GNNs, with regards to their capacity to capture LRIs.

5.2 Limitations

Despite the contributions of this work, there are several limitations to consider.

Theoretical Analysis and Learning Trajectory: As a regularization technique, DropEdge has a significant effect on model convergence (Section 2.3). However, our theoretical analysis does not account for its impact on learning. Understanding how `DropEdge` influences the intermediate stages of learning would provide a more comprehensive view of its effects.

Scope of Theoretical Analysis: Our exact theoretical investigation is limited to linear `GCNs` (Section 3.2.2). Although we extended the existing sensitivity bounds for nonlinear `MPNNs` to the `DropEdge` setting (Section 3.2.3), the use of inequality bounds introduces challenges in drawing reliable and generalizable conclusions. Further theoretical exploration across various nonlinear models could yield more robust insights.

Experimental Models: Our experiments with real-world datasets focused on `GCN` and `GAT` models. While these are representative of commonly used architectures, the results may not be fully applicable to other `GNNs`. Exploring a broader range of models could provide a more comprehensive understanding of `DropEdge`'s effects.

Dataset Scale: Our empirical analysis in Chapter 4 relies on *small-scale* (note, *not* small-range) datasets. The conclusions drawn from these limited datasets may not fully represent the complexities in more extensive real-world scenarios. Larger and more diverse datasets (Section 2.5.4) would have enhanced the reliability and generalizability of our findings.

5.3 Future Directions and Final Remarks

This thesis provides a foundation for understanding the effects of `DropEdge` on `GNNs`, but several promising avenues for future research remain.

Analysis of Other Algorithms for Training Deep GNNs: While this work focuses on `DropEdge`, there is a need for a broader investigation into other methods designed to train deeper `GNNs` (Section 2.5.1). Analyzing various strategies designed for mitigating over-smoothing, particularly in the context of over-squashing, could be invaluable for designing deep `GNNs` for long-range tasks.

Investigation of DropEdge-like Methods: Future research could also consider other `DropEdge`-like techniques that operate on the adjacency matrix. These include DropNode

[Fen+20], DropGNN [Pap+21], and DropAGG [Jia+23]. A thorough examination of these techniques, including their impact on model training and performance, would contribute to a comprehensive understanding of the effect of random edge-dropping on learning LRIs.

Evaluation with Long-Range Datasets: Our study used small-scale datasets, which may limit the generalizability of the findings. Future work should involve a comprehensive evaluation using long-range graph datasets (Section 2.5.4) to better assess how methods like `DropEdge` perform in more complex scenarios. This would provide a deeper understanding of how different techniques handle long-range dependencies and their practical implications for real-world applications.

We hope our work highlights the gap in our understanding of LRI-modelling and encourages the research community to explore the suitability of deep GNNs for long-range tasks. We re-iterate: *the ability of some algorithms to improve the trainability of deep `GNNs` by mitigating issues like vanishing gradients and over-smoothing does not necessarily imply that these models can effectively capture `LRIs`.* To conclude that, we need better theoretical understanding of these algorithms, as well as extensive evaluation on long-range graph benchmarks.

Bibliography

[DS84] Peter G. Doyle and J. Laurie Snell. *Random Walks and Electric Networks.* Carus Mathematical Monographs. Mathematical Association of America, 1984.

[Cha+89] Ashok K. Chandra et al. “The electrical resistance of a graph captures its commute and cover times”. In: *computational complexity* 6 (1989), pp. 312–340.

[LeC+89] Yann LeCun et al. “Handwritten Digit Recognition with a Back-Propagation Network”. In: *Advances in Neural Information Processing Systems.* Ed. by D. Touretzky. Vol. 2. Morgan-Kaufmann, 1989.

[Lov93] L. Lovász. “Random walks on graphs: A survey”. In: *Combinatorics, Paul Erdos is Eighty* 2.1 (1993), pp. 1–46.

[Tau95] Gabriel Taubin. “A signal processing approach to fair surface design”. In: *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques.* SIGGRAPH ’95. New York, NY, USA: Association for Computing Machinery, 1995, pp. 351–358.

[GBL98] C. Lee Giles, Kurt D. Bollacker, and Steve Lawrence. “CiteSeer: an automatic citation indexing system”. In: *Proceedings of the Third ACM Conference on Digital Libraries.* DL ’98. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 1998, pp. 89–98.

[McC+00] Andrew Kachites McCallum et al. “Automating the Construction of Internet Portals with Machine Learning”. In: *Information Retrieval* 3.2 (July 2000), pp. 127–163.

[DD03] Paul D. Dobson and Andrew J. Doig. “Distinguishing Enzyme Structures from Non-enzymes Without Alignments”. In: *Journal of Molecular Biology* 330.4 (2003), pp. 771–783.

[YFW03] J.S. Yedidia, W.T. Freeman, and Y. Weiss. “Understanding Belief Propagation and Its Generalizations”. In: *Exploring Artificial Intelligence in the New Millennium*. Ed. by G. Lakemeyer and B. Nebel. Morgan Kaufmann Publishers, Jan. 2003. Chap. 8, pp. 239–236.

[KMB05] Jeroen Kazius, Ross McGuire, and Roberta Bursi. “Derivation and Validation of Toxicophores for Mutagenicity Prediction”. In: *Journal of Medicinal Chemistry* 48.1 (Jan. 2005), pp. 312–320.

[MS05] Peter Mahlmann and Christian Schindelhauer. “Peer-to-peer networks based on random transformations of connected regular undirected graphs”. In: *Proceedings of the Seventeenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*. SPAA ’05. Las Vegas, Nevada, USA: Association for Computing Machinery, 2005, pp. 155–164.

[Fed+06] Tomas Feder et al. “A Local Switch Markov Chain on Given Degree Graphs with Application in Connectivity of Peer-to-Peer Networks”. In: *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS’06)*. 2006, pp. 69–76.

[WK06] Nikil Wale and George Karypis. “Comparison of Descriptor Spaces for Chemical Compound Retrieval and Classification”. In: *Sixth International Conference on Data Mining (ICDM’06)*. 2006, pp. 678–689.

[Sen+08] Prithviraj Sen et al. “Collective Classification in Network Data”. In: *AI Magazine* 29.3 (Sept. 2008), p. 93.

[Oll09] Yann Ollivier. “Ricci curvature of Markov chains on metric spaces”. In: *Journal of Functional Analysis* 256.3 (2009), pp. 810–864.

[Sca+09] Franco Scarselli et al. “The Graph Neural Network Model”. In: *IEEE Transactions on Neural Networks* 20.1 (2009), pp. 61–80.

[Eve+10] Mark Everingham et al. “The Pascal Visual Object Classes (VOC) Challenge”. In: *International Journal of Computer Vision* 88.2 (June 2010), pp. 303–338.

[Irw+12] John J Irwin et al. “ZINC: a free tool to discover chemistry for biology”. en. In: *J Chem Inf Model* 52.7 (June 2012), pp. 1757–1768.

[LM12] Jure Leskovec and Julian Mcauley. “Learning to Discover Social Circles in Ego Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012.

[Nam+12] Galileo Mark Namata et al. “Query-Driven Active Surveying for Collective Classification”. In: *International Workshop on Mining and Learning with Graphs*. Edinburgh, Scotland: MLG, 2012.

[TMP12] Amanda L. Traud, Peter J. Mucha, and Mason A. Porter. “Social structure of Facebook networks”. In: *Physica A: Statistical Mechanics and its Applications* 391.16 (2012), pp. 4165–4180.

[MHN13] A.L. Maas, A.Y. Hannun, and A.Y. Ng. “Rectifier Nonlinearities Improve Neural Network Acoustic Models”. In: *Proceedings of the International Conference on Machine Learning*. Atlanta, Georgia, 2013.

[LK14] Jure Leskovec and Andrej Krevl. *SNAP Datasets: Stanford Large Network Dataset Collection*. http://snap.stanford.edu/data. June 2014.

[Lin+14] Tsung-Yi Lin et al. “Microsoft COCO: Common Objects in Context”. In: *Computer Vision – ECCV 2014*. Ed. by David Fleet et al. Cham: Springer International Publishing, 2014, pp. 740–755.

[Ram+14] Raghunathan Ramakrishnan et al. “Quantum chemistry structures and properties of 134 kilo molecules”. In: *Scientific Data* 1 (2014).

[Sri+14] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958.

[KB15] Diederik Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations (ICLR)*. San Diega, CA, USA, 2015.

[McA+15] Julian McAuley et al. “Image-Based Recommendations on Styles and Substitutes”. In: *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’15. Santiago, Chile: Association for Computing Machinery, 2015, pp. 43–52.

[Sin+15] Sandeep Singh et al. “SATPdb: a database of structurally annotated therapeutic peptides”. en. In: *Nucleic Acids Res* 44.D1 (Nov. 2015), pp. D1119–26.

[Sze+15] Christian Szegedy et al. “Going deeper with convolutions”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 1–9.

[All+16] Zeyuan Allen-Zhu et al. “Expanders via local edge flips”. In: *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA ’16. Arlington, Virginia: Society for Industrial and Applied Mathematics, 2016, pp. 259–269.

[GG16] Yarin Gal and Zoubin Ghahramani. “Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning”. In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, June 2016, pp. 1050–1059.

[He+16] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778.

[Kaw16] Kenji Kawaguchi. “Deep Learning without Poor Local Minima”. In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee et al. Vol. 29. Curran Associates, Inc., 2016.

[Li+16] Yujia Li et al. “Gated Graph Sequence Neural Networks”. In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2016.

[YCS16] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. “Revisiting Semi-Supervised Learning with Graph Embeddings”. In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, June 2016, pp. 40–48.

[Gil+17] Justin Gilmer et al. “Neural Message Passing for Quantum Chemistry”. In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, Aug. 2017, pp. 1263–1272.

[HYL17] Will Hamilton, Zhitao Ying, and Jure Leskovec. “Inductive Representation Learning on Large Graphs”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017.

[KW17] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *International Conference on Learning Representations (ICLR)*. 2017.

[MBB17] Federico Monti, Michael Bronstein, and Xavier Bresson. "Geometric Matrix Completion with Recurrent Multi-Graph Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017.

[Sch+17] Michael Schlichtkrull et al. *Modeling Relational Data with Graph Convolutional Networks*. 2017.

[Vas+17] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017.

[ZL17] Marinka Zitnik and Jure Leskovec. "Predicting multicellular function through multi-layer tissue networks". In: *Bioinformatics* 33.14 (July 2017), pp. i190–i198.

[CZS18] Jianfei Chen, Jun Zhu, and Le Song. "Stochastic Training of Graph Convolutional Networks with Variance Reduction". In: *International Conference on Machine Learning*. 2018, pp. 941–949.

[GWJ18] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. "Large-Scale Learnable Graph Convolutional Networks". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM. 2018, pp. 1416–1424.

[LHW18] Qimai Li, Zhichao Han, and Xiao-ming Wu. "Deeper Insights Into Graph Convolutional Networks for Semi-Supervised Learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 32.1 (Apr. 2018).

[MDS18] Matthew K Matlock, Na Le Dang, and S Joshua Swamidass. "Learning a Local-Variable Model of Aromatic and Conjugated Systems". en. In: *ACS Cent Sci* 4.1 (Jan. 2018), pp. 52–62.

[Shc+18] Oleksandr Shchur et al. "Pitfalls of Graph Neural Network Evaluation". In: *Relational Representation Learning Workshop, NeurIPS 2018* (2018).

[Vel+18] Petar Veličković et al. "Graph Attention Networks". In: *International Conference on Learning Representations*. 2018.

[Xu+18] Keyulu Xu et al. “Representation Learning on Graphs with Jumping Knowledge Networks”. In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, July 2018, pp. 5453–5462.

[Yin+18] Rex Ying et al. “Graph Convolutional Neural Networks for Web-Scale Recommender Systems”. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD ’18. London, United Kingdom: Association for Computing Machinery, 2018, pp. 974–983.

[Coo+19] Colin Cooper et al. “The flip Markov chain for connected regular graphs”. In: *Discrete Applied Mathematics* 254 (2019), pp. 56–79.

[FL19] Matthias Fey and Jan E. Lenssen. “Fast Graph Representation Learning with PyTorch Geometric”. In: *ICLR Workshop on Representation Learning on Graphs and Manifolds*. 2019.

[Li+19] Guohao Li et al. “DeepGCNs: Can GCNs Go As Deep As CNNs?” In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 9266–9275.

[Mat+19] Matthew K. Matlock et al. “Deep learning long-range information in undirected graphs with wave networks”. In: *2019 International Joint Conference on Neural Networks (IJCNN)*. 2019, pp. 1–8.

[SK19] Connor Shorten and Taghi M. Khoshgoftaar. “A survey on Image Data Augmentation for Deep Learning”. In: *Journal of Big Data* 6.1 (July 2019), p. 60.

[Wan+19] Minjie Wang et al. “Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks”. In: *arXiv preprint arXiv:1909.01315* (2019).

[Bar+20] Pablo Barceló et al. “The Logical Expressiveness of Graph Neural Networks”. In: *International Conference on Learning Representations*. 2020.

[CW20a] Chen Cai and Yusu Wang. *A Note on Over-Smoothing for Graph Neural Networks*. 2020.

[CW20b] Chen Cai and Yusu Wang. *A Note on Over-Smoothing for Graph Neural Networks*. 2020.

[Che+20a] Deli Chen et al. "Measuring and Relieving the Over-Smoothing Problem for Graph Neural Networks from the Topological View". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.04 (Apr. 2020), pp. 3438–3445.

[Che+20b] Ming Chen et al. "Simple and Deep Graph Convolutional Networks". In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, July 2020, pp. 1725–1735.

[Fen+20] Wenzheng Feng et al. "Graph Random Neural Networks for Semi-Supervised Learning on Graphs". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 22092–22103.

[GK20] Hossein Gholamalinezhad and Hossein Khosravi. *Pooling Methods in Deep Neural Networks, a Review*. 2020.

[Gon+20] Shunwang Gong et al. "Geometrically Principled Connections in Graph Neural Networks". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 11412–11421.

[Has+20] Arman Hasanzadeh et al. *Bayesian Graph Neural Networks with Adaptive Connection Sampling*. 2020.

[Hu+20] Weihua Hu et al. "Open Graph Benchmark: Datasets for Machine Learning on Graphs". In: *arXiv preprint arXiv:2005.00687* (2020).

[Hua+20] Lei Huang et al. *Normalization Techniques in Training DNNs: Methodology, Analysis and Application*. 2020.

[LGJ20] Meng Liu, Hongyang Gao, and Shuiwang Ji. "Towards Deeper Graph Neural Networks". In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM. 2020.

[Mor+20] Christopher Morris et al. "TUDataset: A collection of benchmark datasets for learning with graphs". In: *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*. 2020.

[OS20] Kenta Oono and Taiji Suzuki. "Graph Neural Networks Exponentially Lose Expressive Power for Node Classification". In: *International Conference on Learning Representations*. 2020.

[Ron+20] Yu Rong et al. “DropEdge: Towards Deep Graph Convolutional Networks on Node Classification”. In: *International Conference on Learning Representations*. 2020.

[RS20] Benedek Rozemberczki and Rik Sarkar. “Characteristic Functions on Graphs: Birds of a Feather, from Statistical Descriptors to Parametric Models”. In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. CIKM ’20. Virtual Event, Ireland: Association for Computing Machinery, 2020, pp. 1325–1334.

[SGB20] Kimberly Stachenfeld, Jonathan Godwin, and Peter Battaglia. *Graph Networks with Spectral Message Passing*. 2020.

[You+20a] Yuning You et al. “Graph Contrastive Learning with Augmentations”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 5812–5823.

[You+20b] Yuning You et al. “L2-GCN: Layer-Wise and Learned Efficient Training of Graph Convolutional Networks”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 2124–2132.

[You+20c] Yuning You et al. “When Does Self-Supervision Help Graph Convolutional Networks?” In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, July 2020, pp. 10871–10880.

[ZA20] Lingxiao Zhao and Leman Akoglu. “PairNorm: Tackling Oversmoothing in GNNs”. In: *International Conference on Learning Representations*. 2020.

[Zho+20] Kaixiong Zhou et al. “Towards Deeper Graph Neural Networks with Differentiable Group Normalization”. In: *Advances in neural information processing systems*. 2020.

[Zhu+20] Jiong Zhu et al. “Beyond Homophily in Graph Neural Networks: Current Limitations and Effective Designs”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 7793–7804.

[AY21] Uri Alon and Eran Yahav. “On the Bottleneck of Graph Neural Networks and its Practical Implications”. In: *International Conference on Learning Representations*. 2021.

[Bod+21] Cristian Bodnar et al. “Weisfeiler and Lehman Go Cellular: CW Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., 2021, pp. 2625–2640.

[Hu+21] Weihua Hu et al. “OGB-LSC: A Large-Scale Challenge for Machine Learning on Graphs”. In: *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*. Ed. by J. Vanschoren and S. Yeung. Vol. 1. 2021.

[Lim+21] Derek Lim et al. “New Benchmarks for Learning on Non-Homophilous Graphs”. In: *arXiv preprint arXiv:2104.01404* (2021).

[Muk+21] Arjun Mukherjee et al. “What Yelp Fake Review Filter Might Be Doing?” In: *Proceedings of the International AAAI Conference on Web and Social Media* 7.1 (Aug. 2021), pp. 409–418.

[Pap+21] Pál András Papp et al. “DropGNN: Random Dropouts Increase the Expressiveness of Graph Neural Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., 2021, pp. 21997–22009.

[RW21] Ladislav Rampášek and Guy Wolf. “Hierarchical Graph Neural Nets can Capture Long-Range Interactions”. In: *2021 IEEE 31st International Workshop on Machine Learning for Signal Processing (MLSP)*. 2021, pp. 1–6.

[RAS21] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. “Multi-Scale Attributed Node Embedding”. In: *Journal of Complex Networks* 9.2 (2021).

[Zho+21a] Kaixiong Zhou et al. “Dirichlet energy constrained learning for deep graph neural networks”. In: *Advances in neural information processing systems* (2021).

[Zho+21b] Kuangqi Zhou et al. “Understanding and Resolving Performance Degradation in Deep Graph Convolutional Networks”. In: *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 2021, pp. 2728–2737.

[Arn+22] Adrián Arnaiz-Rodríguez et al. “DiffWire: Inductive Graph Rewiring via the Lovász Bound”. In: *Proceedings of the First Learning on Graphs Conference*. Ed. by Bastian Rieck and Razvan Pascanu. Vol. 198. Proceedings of Machine Learning Research. PMLR, Dec. 2022, 15:1–15:27.

[Ban+22] Pradeep Kr. Banerjee et al. “Oversquashing in GNNs through the lens of information contraction and graph expansion”. In: *2022 58th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. Monticello, IL, USA: IEEE Press, 2022, pp. 1–8.

[DLV22] Andreea Deac, Marc Lackenby, and Petar Veličković. “Expander Graph Propagation”. In: *NeurIPS 2022 Workshop: New Frontiers in Graph Learning*. 2022.

[Gia22] George Giakkoupis. “Expanders via local edge flips in quasilinear time”. In: *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2022. Rome, Italy: Association for Computing Machinery, 2022, pp. 64–76.

[Top+22] Jake Topping et al. “Understanding over-squashing and bottlenecks on graphs via curvature”. In: *International Conference on Learning Representations*. 2022.

[You+22] Yuning You et al. “Bringing Your Own View: Graph Contrastive Learning without Prefabricated Data Augmentations”. In: *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*. WSDM ’22. Virtual Event, AZ, USA: Association for Computing Machinery, 2022, pp. 1300–1309.

[Zhe+22] Wenqing Zheng et al. “Cold Brew: Distilling Graph Node Representations with Incomplete or Missing Neighborhoods”. In: *International Conference on Learning Representations*. 2022.

[Bla+23] Mitchell Black et al. “Understanding Oversquashing in GNNs through the Lens of Effective Resistance”. In: *Proceedings of the 40th International Conference on Machine Learning*. Ed. by Andreas Krause et al. Vol. 202. Proceedings of Machine Learning Research. PMLR, July 2023, pp. 2528–2547.

[Di +23] Francesco Di Giovanni et al. “On over-squashing in message passing neural networks: The impact of width, depth, and topology”. In: *International Conference on Machine Learning*. PMLR. 2023, pp. 7865–7885.

[Dwi+23] Vijay Prakash Dwivedi et al. “Benchmarking Graph Neural Networks”. In: *Journal of Machine Learning Research* 24.43 (2023), pp. 1–48.

[Fan+23] Taoran Fang et al. “DropMessage: Unifying Random Dropping for Graph Neural Networks”. In: (2023).

[Fer+23] Oleksandr Ferludin et al. “TF-GNN: Graph Neural Networks in TensorFlow”. In: *CoRR* abs/2207.03522 (2023).

[GYS23] Rickard Brüel Gabrielsson, Mikhail Yurochkin, and Justin Solomon. “Rewiring with Positional Encodings for Graph Neural Networks”. In: *Transactions on Machine Learning Research* (2023).

[Gir+23] Jhony H. Giraldo et al. “On the Trade-off between Over-smoothing and Over-squashing in Deep Graph Neural Networks”. In: *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*. CIKM ’23. Birmingham, United Kingdom: Association for Computing Machinery, 2023, pp. 566–576.

[Gut+23] Benjamin Gutteridge et al. “DRew: Dynamically Rewired Message Passing with Delay”. In: *International Conference on Machine Learning*. PMLR. 2023, pp. 12252–12267.

[Han+23] Jiaqi Han et al. “Structure-Aware DropEdge Toward Deep Graph Convolutional Networks”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2023), pp. 1–13.

[Jia+23] Bo Jiang et al. “DropAGG: Robust Graph Neural Networks via Drop Aggregation”. In: *Neural Networks* 163 (2023), pp. 65–74.

[KBM23] Kedar Karhadkar, Pradeep Kr. Banerjee, and Guido Montufar. “FoSR: First-order spectral rewiring for addressing oversquashing in GNNs”. In: *The Eleventh International Conference on Learning Representations*. 2023.

[Liu+23] Yang Liu et al. “CurvDrop: A Ricci Curvature Based Approach to Prevent Graph Neural Networks from Over-Smoothing and Over-Squashing”. In: *Proceedings of the ACM Web Conference 2023*. WWW ’23. Austin, TX, USA: Association for Computing Machinery, 2023, pp. 221–230.

[Ngu+23] Khang Nguyen et al. “Revisiting over-smoothing and over-squashing using ollivier-ricci curvature”. In: *Proceedings of the 40th International Conference on Machine Learning*. ICML’23. Honolulu, Hawaii, USA: JMLR.org, 2023.

[RBM23] T. Konstantin Rusch, Michael M. Bronstein, and Siddhartha Mishra. *A Survey on Oversmoothing in Graph Neural Networks*. 2023.

[XZL23] Han Xuanyuan, Tianxiang Zhao, and Dongsheng Luo. “Shedding Light on Random Dropping and Oversmoothing”. In: *NeurIPS 2023 Workshop: New Frontiers in Graph Learning*. 2023.

[Gio+24] Francesco Di Giovanni et al. “How does over-squashing affect the power of GNNs?” In: *Transactions on Machine Learning Research* (2024).

[Qia+24] Chendi Qian et al. “Probabilistically Rewired Message-Passing Neural Networks”. In: *The Twelfth International Conference on Learning Representations*. 2024.