Effects of Dropout on Performance in Long-range Graph Learning Tasks

Jasraj Singh[*]
Independent Researcher

Keyue Jiang
University College London

Brooks Paige
University College London

Laura Toni
University College London

Abstract

Message Passing Neural Networks (MPNNs) are a class of Graph Neural Networks (GNNs) that propagate information across the graph via local neighborhoods. The scheme gives rise to two key challenges: *over-smoothing* and *over-squashing*. While several Dropout-style algorithms, such as DropEdge and DropMessage, have successfully addressed over-smoothing, their impact on over-squashing remains largely unexplored. This represents a critical gap in the literature, as failure to mitigate over-squashing would make these methods unsuitable for long-range tasks – the intended use case of deep MPNNs. In this work, we study the aforementioned algorithms, and closely related edge-dropping algorithms – DropNode, DropAgg and DropGNN – in the context of over-squashing. We present theoretical results showing that DropEdge-variants reduce sensitivity between distant nodes, limiting their suitability for long-range tasks. To address this, we introduce DropSens, a sensitivity-aware variant of DropEdge that explicitly controls the proportion of information lost due to edge-dropping, thereby increasing sensitivity to distant nodes despite dropping the same number of edges. Our experiments on long-range synthetic and real-world datasets confirm the predicted limitations of existing edge-dropping and feature-dropping methods. Moreover, DropSens consistently outperforms graph rewiring techniques designed to mitigate over-squashing, suggesting that simple, targeted modifications can substantially improve a model's ability to capture long-range interactions. Our conclusions highlight the need to re-evaluate and re-design existing methods for training deep GNNs, with a renewed focus on modelling long-range interactions.

1 Introduction

Graph neural networks (GNNs) [51, 71] are powerful neural models developed for modelling graph-structured data, and have found applications in several real-world scenarios [29, 60, 82, 88–92, 94, 99]. A popular class of GNNs, called *message-passing neural networks* (MPNNs) [32], recursively process neighborhood information using message-passing layers. These layers are stacked to allow each node to aggregate information from increasingly larger neighborhoods, akin to how *convolutional neural networks* (CNNs) learn hierarchical features for images [48]. However, unlike in image-based deep learning, where *ultra-deep* CNN architectures have led to performance breakthroughs [38, 78], shallow GNNs often outperform deeper models on many graph learning

[*]Part of the work done as a master's student at UCL. Correspondence at jasraj.singh00150@gmail.com.

Preprint. Under review.

tasks [97]. This is because deep GNNs suffer from unique issues like *over-smoothing* [64] and *over-squashing* [4], which makes training them notoriously difficult.

Over-smoothing refers to the problem of node representations becoming *too similar* as they are recursively processed. This is undesirable since it limits the GNN from effectively utilizing the information in the input features. The problem has garnered significant attention from the research community, resulting in a suite of algorithms designed to address it [70] (see Appendix A.1 for an overview of representative methods). Amongst these methods are a collection of random edge-dropping algorithms, including DropEdge [68], DropNode [23], DropAgg [43] and DropGNN [65] – which we will collectively refer to as *DropEdge-variants* – which act as *message-passing reducers*. In addition, we have DropMessage [21], which performs Dropout [77] on the message matrices, instead of the feature matrices; we will collectively refer to these two methods as *Dropout-variants* since they are applied along the feature dimensions.

The other issue specific to GNNs is over-squashing. In certain graph structures, neighborhood size grows exponentially with distance from the source [12], causing information to be lost as it passes through graph bottlenecks [4]. This limits MPNNs' ability to enable communication between distant nodes, which is crucial for good performance on long-range tasks. To alleviate over-squashing, several graph-rewiring techniques have been proposed, which aim to improve graph connectivity by adding edges in a strategic manner [4, 8, 16, 44, 63] (see Appendix A.3 for an overview of representative methods).[2] In contrast, the DropEdge-variants only remove edges, which should, in principle, amplify over-squashing levels. The same can be intuitively argued about Dropout-variants.

Empirical evidence in support of methods designed for training deep GNNs has been majorly collected on short-range tasks (see Appendix A.2 for a detailed discussion). That is, it simply suggests that *these methods prevent loss of local information, but it remains inconclusive if they facilitate capturing long-range interactions* (LRIs). Of course, on long-range tasks, deep GNNs are useless if they cannot capture LRIs. This is especially a concern for DropEdge-variants since evidence suggests that alleviating over-smoothing with graph rewiring could exacerbate over-squashing [34, 63].

Contributions. In this work, we precisely characterize the effects of random edge-dropping algorithms on over-squashing in MPNNs. By explicitly computing the expected *sensitivity* of the node representations to the node features [79] (inversely related to over-squashing) in a linear Graph Convolutional Network (GCN) [47], we show that these methods provably reduce the *effective receptive field* of the model. Precisely speaking, the rate at which sensitivity between nodes decays is *exponential* w.r.t. the distance between them. We also extend the existing theoretical results on sensitivity in nonlinear MPNNs [8, 18, 84] to the random edge-dropping setting, concluding that these algorithms exacerbate the over-squashing problem. We use our analysis of GCNs to design a sensitivity-aware DropEdge-variant, named *DropSens*, that enjoys the representational expressivity of DropEdge without suffering from over-squashing, thereby demonstrating how algorithms can be readily adapted for long-range tasks.

We evaluated the DropEdge- and Dropout-variants on long-range datasets using GCN, Graph Isomorphism Network (GIN) [85] and Graph Attention Network (GAT) [81] architectures. Specifically, we follow the setup in [33] with the SyntheticZINC dataset, in [79] with real-world homophilic (corresponding to short-range tasks) and heterophilic (long-range tasks) node classification datasets, and in [8, 44] with graph classification datasets. Our results indicate that while the random dropping methods improve model performance in short-range tasks, they are often ineffective, and sometimes even detrimental, to long-range task performance. Finally, we present results for DropSens, which outperforms state-of-the-art graph rewiring methods aimed at addressing over-squashing at node classification and graph-classification tasks. These findings point to the importance of re-evaluating the methods used to train deep GNNs, especially in terms of how well they capture LRIs.

2 Background

Consider a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with $\mathcal{V} = [N] \coloneqq \{1, \dots, N\}$ denoting the node set and $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ the edge set; $(j \to i) \in \mathcal{E}$ if there's an edge from node j to node i. Let $\boldsymbol{A} \in \{0,1\}^{N \times N}$ denote its adjacency matrix, such that $\boldsymbol{A}_{ij} = 1$ if and only if $(j \to i) \in \mathcal{E}$, and let $\boldsymbol{D} \coloneqq \operatorname{diag}(\boldsymbol{A}\mathbf{1}_N)$ denote the in-degree matrix. The geodesic distance, $d_{\mathcal{G}}(j, i)$, from node j to node i is the length

[2] Sometimes, along with removal of some edges to preserve statistical properties of the original topology.

of the shortest path starting at node j and ending at node i. Accordingly, the ℓ-hop neighborhood of a node i can be defined as the set of nodes that can reach it in exactly $\ell \in \mathbb{N}_0$ steps, $\mathbb{S}^{(\ell)}(i) = \{j \in \mathcal{V} : d_{\mathcal{G}}(j,i) = \ell\}$.

2.1 Graph Neural Networks

Graph Neural Networks (GNNs) operate on inputs of the form $(\mathcal{G}, \boldsymbol{X})$, where $\mathcal{G}$ encodes the graph topology and $\boldsymbol{X} \in \mathbb{R}^{N \times H^{(0)}}$ collects the node features.[3] Message-Passing Neural Networks (MPNNs) [32] are a special class of GNNs which recursively aggregate information from the 1-hop neighborhood of each node using *message-passing layers*. An L-layer MPNN is given as

$$\begin{aligned} \boldsymbol{z}_i^{(\ell)} &= \mathsf{Upd}^{(\ell)}\left(\boldsymbol{z}_i^{(\ell-1)}, \mathsf{Agg}^{(\ell)}\left(\boldsymbol{z}_i^{(\ell-1)}, \left\{\boldsymbol{z}_j^{(\ell-1)} : j \in \mathbb{S}^{(1)}(i)\right\}\right)\right), \quad \forall \ell \in [L] \\ \mathsf{MPNN}_\theta(\mathcal{G}, \boldsymbol{X}) &= \left\{\mathsf{Out}\left(\boldsymbol{z}_i^{(L)}\right) : i \in \mathcal{V}\right\} \end{aligned} \tag{2.1}$$

where $\boldsymbol{Z}^{(0)} = \boldsymbol{X}$, $\mathsf{Agg}^{(\ell)}$ denotes the *aggregation functions*, $\mathsf{Upd}^{(\ell)}$ the *update functions*, and Out the *readout function*. Since $\boldsymbol{z}_i^{(L)}$ is a function of the input features of nodes at most L-hops away from it, its *receptive field* is given by $\mathbb{B}^{(L)}(i) \coloneqq \{j \in \mathcal{V} : d_{\mathcal{G}}(j,i) \leq L\}$.

For example, a GCN [47] updates node representations as the weighted sum of its neighbors' representations:

$$\boldsymbol{Z}^{(\ell)} = \sigma\left(\hat{\boldsymbol{A}}\boldsymbol{Z}^{(\ell-1)}\boldsymbol{W}^{(\ell)}\right) \tag{2.2}$$

where σ is a point-wise nonlinearity, e.g. ReLU, the propagation matrix, $\hat{\boldsymbol{A}}$, is a *graph shift operator*, i.e. $\hat{\boldsymbol{A}}_{ij} \neq 0$ if and only if $(j \to i) \in \mathcal{E}$ or $i = j$, and $\boldsymbol{W}^{(\ell)} \in \mathbb{R}^{H^{(\ell-1)} \times H^{(\ell)}}$ is a weight matrix. The original choice for $\hat{\boldsymbol{A}}$ was the symmetrically normalized adjacency matrix $\hat{\boldsymbol{A}}^{\text{sym}} \coloneqq \widetilde{\boldsymbol{D}}^{-1/2}\widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{D}}^{-1/2}$ [47], where $\widetilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}_N$ and $\widetilde{\boldsymbol{D}} = \text{diag}(\widetilde{\boldsymbol{A}}\boldsymbol{1}_N)$. However, several influential works have also used the asymmetrically normalized adjacency, $\hat{\boldsymbol{A}}^{\text{asym}} \coloneqq \widetilde{\boldsymbol{D}}^{-1}\widetilde{\boldsymbol{A}}$ [36, 50, 72].

2.2 DropEdge-variants

DropEdge [68] is a random data augmentation technique that works by sampling a subgraph of the input graph in each layer, followed by the addition of self-loops, and uses that for message passing. Several variants of DropEdge have also been proposed, forming a family of random edge-dropping algorithms for tackling the over-smoothing problem. For example, DropNode [23] independently samples nodes and sets their features to $\boldsymbol{0}$, followed by rescaling to make the feature matrix unbiased. This is equivalent to setting the corresponding columns of the propagation matrix to $\boldsymbol{0}$. In a similar vein, DropAgg [43] samples nodes that don't aggregate messages from their neighbors. This is equivalent to dropping the corresponding rows of the adjacency matrix. Combining these two approaches, DropGNN [65] samples nodes which neither propagate nor aggregate messages in a given layer. These algorithms alleviate over-smoothing by reducing the number of messages being propagated in the graph, thereby slowing down the convergence of node representations.

2.3 Dropout-variants

Dropout is a stochastic regularization technique which reduces over-fitting by randomly dropping features before each layer. It has been successful with various architectures, like CNNs [77] and transformers [80], and has also found applications in GNN training. DropMessage [21] is a variant of Dropout designed specifically for message-passing schemes – it acts directly on the messages over each edge, instead of the node representations. This reduces the induced variance in the messages compared to Dropout, DropEdge and DropNode, while at the same time making the method more effective at alleviating over-smoothing and enabling the training of deep GNNs.

[3] To keep things simple, we will ignore edge features.

2.4 Over-squashing

Over-squashing refers to the problem of information from exponentially growing neighborhoods [11] being squashed into finite-sized node representations [4]. [79] formally characterized over-squashing in terms of the Jacobian of the node-level representations w.r.t. the input features: $\|\partial \boldsymbol{z}_i^{(L)}/\partial \boldsymbol{x}_j\|_1$. Accordingly, over-squashing 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.

See Appendix A for an extensive discussion of related works addressing the problems of over-smoothing and over-squashing, and a unified treatment of the two.

3 Sensitivity Analysis

In this section, we perform a theoretical analysis of the expectation – w.r.t. random edge masks – of sensitivity of node representations. This will allow us to predict how DropEdge-variants affect communication between nodes at various distances, which is relevant for predicting their suitability towards learning LRIs.

Here, we present our analysis for linear GCNs, and treat more general nonlinear MPNN architectures in Appendix C.1. In this model, the final node representations can be summarised as

$$\boldsymbol{Z}^{(L)} = \left(\prod_{\ell=1}^{L} \hat{\boldsymbol{A}}^{(\ell)}\right) \boldsymbol{X}\boldsymbol{W} \in \mathbb{R}^{N \times H^{(L)}} \tag{3.1}$$

where $\boldsymbol{W} \coloneqq \prod_{\ell=1}^{L} \boldsymbol{W}^{(\ell)} \in \mathbb{R}^{H^{(0)} \times H^{(L)}}$. Using the i.i.d. assumption on the distribution of edge masks in each layer, the expected sensitivity of node i to node j can be shown to be

$$\mathbb{E}_{\mathbf{M}^{(1)},\dots,\mathbf{M}^{(L)}}\left[\left\|\frac{\partial \boldsymbol{z}_i^{(L)}}{\partial \boldsymbol{x}_j}\right\|_1\right] = \left(\mathbb{E}\left[\hat{\boldsymbol{A}}\right]^L\right)_{ij} \|\boldsymbol{W}\|_1 \tag{3.2}$$

To keep things simple, we will ignore the effect of DropEdge-variants on the optimization trajectory. Accordingly, it is sufficient to study $\mathbb{E}[\hat{\boldsymbol{A}}]$ in order to predict their effect on over-squashing. To maintain analytical tractability, we assume the use of an asymmetrically normalized adjacency matrix for message-passing, $\hat{\boldsymbol{A}} = \hat{\boldsymbol{A}}^{\text{asym}}$.

Lemma 3.1. *The expected propagation matrix under DropEdge is given as:*

$$\begin{aligned} \dot{\boldsymbol{P}}_{ii} &\coloneqq \mathbb{E}_{\mathsf{DE}}\left[\hat{\boldsymbol{A}}_{ii}\right] = \frac{1-q^{d_i+1}}{(1-q)\,(d_i+1)} \\ \dot{\boldsymbol{P}}_{ij} &\coloneqq \mathbb{E}_{\mathsf{DE}}\left[\hat{\boldsymbol{A}}_{ij}\right] = \frac{1}{d_i}\left(1 - \frac{1-q^{d_i+1}}{(1-q)\,(d_i+1)}\right) \end{aligned} \tag{3.3}$$

where $q \in [0,1)$ is the dropping probability.

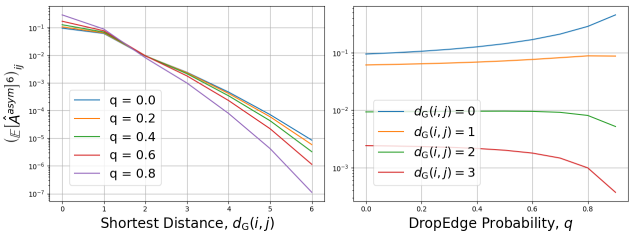
See Appendix B.1 for a proof, and a similar treatment of DropNode, DropAgg and DropGNN.

1-Layer Linear GCNs. $\forall q \in (0,1)$ we have

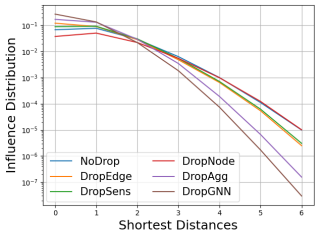
$$\begin{aligned} \dot{\boldsymbol{P}}_{ii} &= \frac{1}{d_i+1}\sum_{k=0}^{d_i} q^k > \frac{1}{d_i+1} \\ \dot{\boldsymbol{P}}_{ij} &= \frac{1}{d_i}\left(1 - \dot{\boldsymbol{P}}_{ii}\right) < \frac{1}{d_i+1} \end{aligned} \tag{3.4}$$

where the right-hand sides of the two inequalities are the corresponding entries in the propagation matrix of a NoDrop model. Equations 3.3, 3.4, B.16 and B.17 together imply the following result:

Lemma 3.2. *In a 1-layer linear GCN with $\hat{\boldsymbol{A}} = \hat{\boldsymbol{A}}^{asym}$, using DropEdge, DropAgg or DropGNN*



(a) Entries of $\dot{\boldsymbol{P}}^6$ decay at exponential rate w.r.t. distance between nodes, and polynomial rate w.r.t. to the DropEdge probability.



(b) MC-approximation of influence distribution in ReLU-GCNs.

Figure 1: Empirical sensitivity analysis using the Cora dataset.

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

2. *decreases the sensitivity to its neighbors' features.*

L-layer Linear GCNs. Unfortunately, we cannot draw similar conclusions in L-layer networks, for nodes at arbitrary distances. To see this, view $\dot{\boldsymbol{P}}$ as the transition matrix of a non-uniform random walk. This walk has higher self-transition ($i = j$) probabilities than in a uniform augmented random walk ($\boldsymbol{P} = \hat{\boldsymbol{A}}^{\text{asym}}$, $q = 0$), but lower inter-node ($i \neq j$) transition probabilities. Note that $\dot{\boldsymbol{P}}^L$ and $\boldsymbol{P}^L$ store the L-step transition probabilities in the corresponding walks. Then, since the paths connecting the nodes $i \in \mathcal{V}$ and $j \in \mathbb{B}^{(L-1)}(i)$ may involve self-loops, $(\dot{\boldsymbol{P}}^L)_{ij}$ may be lower or higher than $(\boldsymbol{P}^L)_{ij}$. Therefore, we cannot conclude how sensitivity between nodes separated by at most $L-1$ hops changes. For nodes L-hops away, however, we can show that DropEdge always decreases the corresponding entry in $\dot{\boldsymbol{P}}^L$, reducing the effective reachability of GCNs. Using Equations B.16 and B.17, we can show the same for DropAgg and DropGNN, respectively.

Theorem 3.1. *In an L-layer linear GCN with $\hat{\boldsymbol{A}} = \hat{\boldsymbol{A}}^{asym}$, using DropEdge, DropAgg or DropGNN decreases the sensitivity of a node $i \in \mathcal{V}$ to another node $j \in \mathbb{S}^{(L)}(i)$, thereby reducing its effective receptive field. Moreover, the sensitivity decreases with increasing dropping probability.*

See Appendix B.2 for a precise quantitative statement and the proof.

Nodes at Arbitrary Distances. Although no general statement could be made about the change in sensitivity between nodes up to $L-1$ hops away, we can analyze such pairs empirically. We compute the L-hop transition matrix $\dot{\boldsymbol{P}}^L$ – proportional to expected sensitivity in linear GCNs under DropEdge – for the Cora dataset, and average the entries after binning node pairs by the shortest distance between them. The results are shown in Figure 1a. In the left subfigure, we observe that the expected sensitivity decays at an *exponential rate* with increasing distance between the corresponding nodes. In the middle subfigure, 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. Similar conclusions can be made with the symmetrically normalized propagation matrix (see Appendix D.1). Note that the over-squashing effects of DropAgg and DropGNN would, in theory, be even more severe, as suggested by Equations B.16 and B.17.

Nonlinear MPNNs. While linear networks are useful in simplifying the theoretical analysis, they are often not practical. In Appendix C.1, we treat the upper bounds on sensitivity established in previous works, and extend them to the DropEdge-variants. Even still, although theoretical bounds offer valuable guarantees, they can be arbitrarily loose in the absence of error quantification, making their practical relevance unclear. To reliably conclude the empirical behaviour of DropEdge- and Dropout-variants, we turn to Monte Carlo simulations with ReLU-GCNs; see Appendix D.3 for a description of the experiment setup. Figure 1b compares the influence of the source nodes [84] at different distances using a dropout probability of 0.5. We observe that while the effect of DropNode on the sensitivity profile – as compared to the baseline NoDrop – is relatively insignificant, models using DropEdge, DropAgg and DropGNN have remarkably lower sensitivity to distant nodes, as predicted by our theory.

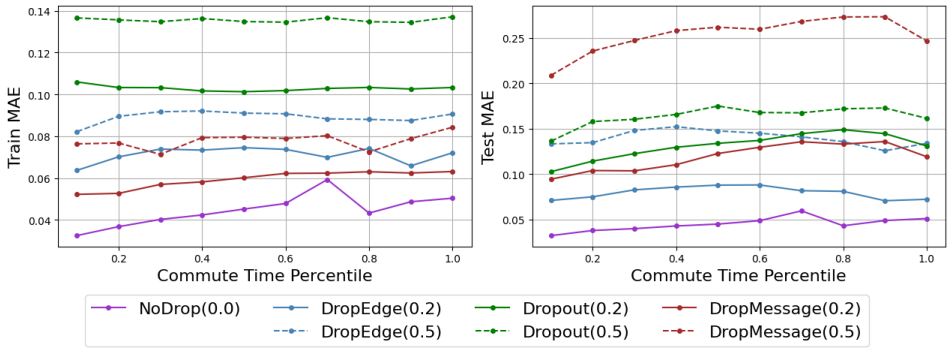


Figure 2: Train and test MAE of 11-layer GCNs on the SyntheticZINC dataset, averaged over 10 initializations.

4 Sensitivity-Aware DropEdge

Lemma 3.1 tells us that DropEdge decreases the weight of cross-edges, $(j \to i)$, in the expected propagation matrix, i.e. the *strength of message passing* over these edges decreases. The fraction of *information preserved* over a cross-edge is dependent only on the dropping probability and the target node's in-degree, d_i. We can directly control this quantity using a per-edge dropping probability, q_i, dependent only on the receiving node's in-degree:

$$c = \frac{\mathbb{E}_{\mathsf{DE}}[\hat{\boldsymbol{A}}_{ij}]}{\mathbb{E}_{\mathsf{ND}}[\hat{\boldsymbol{A}}_{ij}]} = \frac{d_i + 1}{d_i}\left(1 - \frac{1 - q_i^{d_i+1}}{(1 - q_i)\,(d_i + 1)}\right) \implies 1 - c = \frac{q_i - q_i^{d_i+1}}{d_i\,(1 - q_i)} \tag{4.1}$$

where c is the fraction of information preserved, e.g. 95%. We can solve for q_i and mask the incoming edges to node i accordingly; we name this algorithm *DropSens*. In Appendix E.3, we present a `Python` implementation of the algorithm, as well as a computationally efficient approximation to Equation 4.1. In Figure 1b, we observe that DropSens improves sensitivity between distant nodes, compared to DropEdge.

5 Experiments

Our theoretical analysis indicates that random dropping may degrade the performance of GNNs in tasks that depend on capturing LRIs. In this section, we test this hypothesis by evaluating DropEdge- and Dropout-variants on both synthetic and real-world datasets. A complete description of the datasets is provided in Appendix E.1, and the experimental details are in Appendix E.2.

5.1 Synthetic Datasets

SyntheticZINC [33] is a synthetic variant of the ZINC dataset [42], designed to study the effect of information mixing in graph learning. Node features are sparsely assigned, and the target requires non-linear mixing of two selected nodes' features, chosen based on their commute time [10]. We vary the mixing level and evaluate an 11-layer GCN, ensuring sufficient message passing. For better readability, we only test DropEdge, Dropout and DropMessage – the three more popular methods used for training deep GNNs.

The results are presented in Figure 2, where we can observe that the mean absolute error (MAE) increases with the commute time percentile used to select the node pairs, as was hypothesized and evidenced in [33]. Additionally, we observe that both train and test performance decline when using dropout with a probability as low as 0.2, and even more so with a higher probability of 0.5. These results provide strong evidence for the detrimental effects of dropout methods in modelling long-range interactions, supporting our theoretical analysis.

5.2 Real-world Datasets

To test the dropping methods on real-world datasets, we use the GCN, GIN [85] and GAT [81] architectures – GCN and GIN satisfy the model assumptions made in all the theoretical results pre-

Table 1: Difference in mean test accuracy (%) between the best performing configuration of each dropout method and the baseline NoDrop model. Cell colors represent p-values from a t-test evaluating whether dropout improves performance: green indicates significance at 90% confidence, while red denotes insignificant results.

(a) Node classification tasks.

GNN	Dropout	Homophilic Networks			Heterophilic Networks		
		Cora	CiteSeer	PubMed	Chameleon	Squirrel	TwitchDE
GCN	DropEdge	+0.419	+0.686	+0.385	−0.634	+0.009	−0.093
	DropNode	+0.373	+0.197	+0.841	−0.674	−0.656	−0.113
	DropAgg	+0.224	+0.486	−0.245	−10.640	−13.970	−6.674
	DropGNN	+0.396	+0.820	+0.506	−1.774	−0.291	−0.395
	Dropout	+0.628	+0.128	+1.218	−1.395	−0.110	−0.260
	DropMessage	+0.030	−0.388	+1.217	+1.322	+0.320	+0.160
GIN	DropEdge	−0.276	−0.529	−0.034	−0.726	+0.289	−0.124
	DropNode	+0.379	+0.926	+0.296	−1.832	−0.265	−0.171
	DropAgg	+0.095	−0.254	−0.487	−1.153	+0.275	+0.205
	DropGNN	−0.478	−2.284	−1.366	−1.642	−0.066	−0.008
	Dropout	+1.347	+0.011	+0.368	−2.702	−0.152	−0.573
	DropMessage	+2.602	+0.219	+1.030	+0.943	+0.108	−0.025

(b) Graph classification tasks.

GNN	Dropout	Molecular Networks			Social Networks		
		Mutag	Proteins	Enzymes	Reddit	IMDb	Collab
GCN	DropEdge	−1.100	+1.750	−0.589	−8.380	+1.300	−1.145
	DropNode	−5.100	+2.339	−2.292	−7.440	+2.720	−4.054
	DropAgg	−0.900	+1.214	−0.553	−16.460	+3.580	−30.386
	DropGNN	−0.200	+1.589	−2.918	−11.860	+1.540	+0.705
	Dropout	−0.300	+2.018	−6.208	−6.050	+1.240	−1.729
	DropMessage	+2.000	+2.143	−4.906	−7.360	+1.300	−0.330
GIN	DropEdge	−1.100	−1.804	−4.716	−2.700	−1.820	−0.642
	DropNode	−3.800	−2.911	−0.493	+0.550	−0.120	+1.206
	DropAgg	−3.800	−1.982	−0.640	+0.930	−1.640	−5.943
	DropGNN	−7.000	−2.750	−3.694	+1.210	−5.200	−6.593
	Dropout	−2.400	−3.446	−2.047	+2.590	−1.180	−0.180
	DropMessage	−3.600	−1.125	−0.302	+0.850	−0.460	+1.126

sented in Section 3, while GAT does not satisfy any of them, since the attention scores are computed as a function of *all* the node representations. Therefore, GCN, GIN and GAT together provide a broad representation of different MPNN architectures. We present the results for GCN and GIN in the main text, since these models were used as baselines in a majority of works on alleviating over-squashing [4, 5, 8, 35, 44, 67, 79]; the results for GAT are reported in Table 6.

For each dataset−model−dropout combination, we perform 20 independent runs to find the best performing dropout configuration; results are reported in Table 9. We then perform a t-test to assess whether dropout improves performance, using 50 samples from the NoDrop model ($q = 0$) and 50 samples from the best performing dropout configuration.[4] In this section, we report the p-values of the tests, and in Table 7, we report the effect sizes as Hedges' g statistic [39].

Node-classification. Although determining whether a task requires modelling LRIs can be challenging, understanding the structure of the datasets can provide important insight. For example, homophilic datasets have local consistency in node labels, i.e. nodes closely connected to each other have similar labels. On the other hand, in heterophilic datasets, nearby nodes often have dissimilar labels. Since DropEdge-variants increase the sensitivity of a node's representations to its immediate neighbors, and reduce its sensitivity to distant nodes, we expect it to improve performance on homophilic datasets but harm performance on heterophilic ones; such a setup was also used in [79].

[4]The t-test assumes that both samples are drawn from normal distributions – all Shapiro-Wilk tests for non-normality of samples [74] failed at 90% confidence.

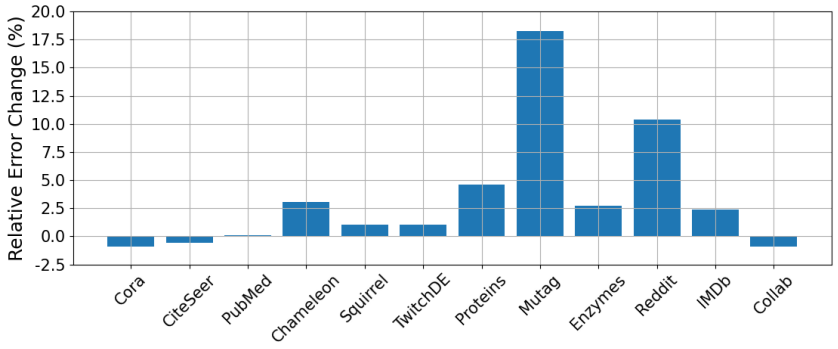


Figure 3: Relative change in test-time performance of a GCN using DropSens, compared to the baseline DropEdge, on real-world datasets from Section 5.2.

In this work, we use Cora [57], CiteSeer [31] and PubMed [62] as representatives of homophilic datasets [52, 98], and Squirrel, Chameleon and TwitchDE [69] to represent heterophilic datasets [52]. The networks' statistics are presented in Table 4, where we can note the remarkably lower homophily measures of heterophilic datasets.

The results are presented in Table 1a. It is clear to see that dropout *significantly improves* test performance on homophilic datasets – with $40/54 \approx 74\%$ cases performing better than the corresponding NoDrop baseline – indicating that these methods are indeed beneficial in tackling short-range tasks. On the other hand, with the heterophilic datasets, the improvement is *insignificant*. Rather, in most ($45/54 \approx 83\%$) cases, the best dropout configuration performs worse than the NoDrop baseline. This suggests that the dropping methods harm generalization in long-range tasks by forcing models to overfit to short-range signals (see Appendix F.3 for supporting evidence).

Graph-classification. Several graph classification datasets have also been identified as long-range tasks, like the molecular networks datasets Mutag [17], Proteins [19] and Enzymes [9], and the social networks datasets Reddit, IMDb and Collab [87]. These datasets have also been used for evaluation in previous works on over-squashing, including [8, 44].

The results are shown in Table 1b, where we observe that dropout methods generally have insignificant effects on model performance, and often even a non-positive effect ($67/108 \approx 62\%$ cases). Notably, the p-values are lower as compared to those recorded for heterophilic datasets in Table 1a, i.e. higher evidence for efficacy of dropping methods. We conjecture that over-squashing may have limited impact on model performance in graph-level tasks since the aggregation module eventually mixes information from distant nodes for computing graph-level representations.

5.3 Evaluating DropSens

We start by comparing DropSens with DropEdge on real-world datasets from Section 5.2, illustrating how algorithms can be readily adapted for better suitability at modelling LRIs. In Figure 3, we present the relative change in error rate ($1 - \text{Acc}$) of DropSens w.r.t. DropEdge, with GCN as the base model. It is clear to observe a uniform improvement in the performance on long-range tasks, suggesting that addressing over-squashing using DropSens can enhance the effectiveness of GCNs.

We now benchmark DropSens against state-of-the-art graph-rewiring techniques designed specifically to tackle over-squashing (see Appendix A.3 for their descriptions). We train a GCN on node classification tasks, following the setup in [79], and both GCN and GIN on graph classification tasks, following [8, 44]. The results with GCN are reported in Table 2a and Table 2b, where we find that DropSens outperforms other methods in node classification tasks, and performs competitively in graph classification tasks. In addition to superior performance, another advantage of DropSens over the other methods is that it significantly reduces the number of messages being propagated, thereby tackling the problem of over-smoothing and increasing training speed.

The results with GIN are presented in Appendix G.3, where we observe that DropSens does not perform competitively – unsurprising, since DropSens was specifically designed to work with GCN's message-passing scheme.

Table 2: Performance of GCN with graph rewiring methods. **First**, **second**, and **third** best results are coloured.

(a) Node-classification tasks – results for other methods taken from [79, Table 2].

Rewiring	Cora	CiteSeer	PubMed	Chameleon	Squirrel	Actor
None	81.89	72.31	78.16	41.33	30.32	23.84
Undirected	-	-	-	42.02	35.53	21.45
+FA	81.65	70.47	**79.48**	42.67	36.86	24.14
DIGL (PPR)	**83.21**	**73.29**	78.84	42.02	33.22	24.77
DIGL + Undirected	-	-	-	42.68	32.48	**25.45**
SDRF	**82.76**	**72.58**	**79.10**	**42.73**	**37.05**	**28.42**
SDRF + Undirected	-	-	-	**44.46**	**37.67**	**28.35**
DropSens	**84.98**	**73.35**	**84.30**	**53.01**	**41.32**	22.38

(b) Graph-classification tasks – results for other methods from [8, Table 1].

Rewiring	Mutag	Proteins	Enzymes	Reddit	IMDb	Collab
None	72.15	70.98	27.67	68.26	**49.77**	33.78
Last FA	70.05	**71.02**	26.47	68.49	48.98	33.32
Every FA	70.45	60.04	18.33	48.49	48.17	**51.80**
DIGL	**79.70**	70.76	**35.72**	**76.04**	**64.39**	**54.50**
SDRF	71.05	70.92	**28.37**	68.62	49.40	33.45
FoSR	**80.00**	**73.42**	25.07	**70.33**	49.66	33.84
GTR	**79.10**	**72.59**	27.52	68.99	**49.92**	33.05
DropSens	70.20	70.61	**36.01**	**77.44**	49.32	**61.68**

6 Conclusion

There exists an important gap in our understanding of several algorithms designed for training deep GNNs – while their positive effects on model performance have been well-studied, making them popular choices for training deep GNNs, their evaluation has been limited to short-range tasks. This is rooted in a key assumption: that if a deep GNN is trainable, it must also be capable of modelling LRIs. As a result, potential adverse effects of these algorithms on capturing LRIs have been overlooked. Our results challenge this assumption – we theoretically and empirically show that DropEdge- and Dropout-variants exacerbate the over-squashing problem in deep GNNs, and degrade performance on long-range tasks. This highlights the need for a more comprehensive evaluation of common training practices for deep GNNs, with special emphasis on their capacity to capture LRIs. This is crucial for building confidence in their use beyond controlled benchmarks.

Limitations. While our theoretical analysis successfully predicts how DropEdge-variants affect test performance on short-range and long-range tasks, it is based on several simplifying assumptions on the message-passing scheme. These assumptions, although standard in the literature, limit the generalizability of our conclusions to other architectures, including ResGCNs [51], GATs [81], and Graph Transformers [83]. Additionally, an important limitation of DropSens is that it requires an architecture-specific alteration to the edge-dropping strategy, which is not practical in general. As also mentioned in Section 4, we did not intend to introduce DropSens as a benchmark, but rather to demonstrate how methods designed for alleviating over-smoothing can be readily adapted to simultaneously control over-squashing.

Future Directions. Currently, real-world datasets are classified as short- or long-range tasks based on extensive model training [4] or weak proxy measures like node homophily [79]. Developing a reliable measure of information mixing in the ground-truth data could greatly benefit the research community. Such a measure would enable more precise identification of short-, intermediate- and long-range tasks, improving evaluation and benchmarking. Another interesting direction is to investigate the significance of over-squashing in graph-level tasks, where the aggregation module of MPNNs enables some mixing of information from distant nodes. To the best of our knowledge, [33] is the only work that directly addresses this question, offering strong theoretical insights. However, empirical validation of these effects remains limited.

References

[1] Zeyuan Allen-Zhu, Aditya Bhaskara, Silvio Lattanzi, Vahab Mirrokni, and Lorenzo Orecchia. Expanders via local edge flips. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, pp. 259–269, USA, 2016. Society for Industrial and Applied Mathematics.

[2] N. Alon and V. D. Milman. $\lambda 1$, isoperimetric inequalities for graphs, and superconcentrators. *Journal of Combinatorial Theory, Series B*, 38(1):73–88, February 1985.

[3] Noga Alon. Eigenvalues and expanders. *Combinatorica*, 6(2):83–96, June 1986.

[4] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations*, 2021.

[5] Adrián Arnaiz-Rodríguez, Ahmed Begga, Francisco Escolano, and Nuria M Oliver. Diffwire: Inductive graph rewiring via the lovász bound. In Bastian Rieck and Razvan Pascanu (eds.), *Proceedings of the First Learning on Graphs Conference*, volume 198 of *Proceedings of Machine Learning Research*, pp. 15:1–15:27. PMLR, 12 2022.

[6] Pradeep Kr. Banerjee, Kedar Karhadkar, Yu Guang Wang, Uri Alon, and Guido Montúfar. Oversquashing in gnns through the lens of information contraction and graph expansion. In *2022 58th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 1–8. IEEE Press, 2022.

[7] Pablo Barceló, Egor V. Kostylev, Mikael Monet, Jorge Pérez, Juan Reutter, and Juan Pablo Silva. The logical expressiveness of graph neural networks. In *International Conference on Learning Representations*, 2020.

[8] Mitchell Black, Zhengchao Wan, Amir Nayyeri, and Yusu Wang. Understanding oversquashing in GNNs through the lens of effective resistance. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 2528–2547. PMLR, 07 2023.

[9] Karsten M. Borgwardt, Cheng Soon Ong, Stefan Schönauer, S. V. N. Vishwanathan, Alex J. Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1):i47–i56, 06 2005.

[10] Ashok K. Chandra, Prabhakar Raghavan, Walter L. Ruzzo, Roman Smolensky, and Prasoon Tiwari. The electrical resistance of a graph captures its commute and cover times. *computational complexity*, 6:312–340, 1989.

[11] Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction. In *International Conference on Machine Learning*, pp. 941–949, 2018.

[12] Jie Chen, Tengfei Ma, and Cao Xiao. FastGCN: Fast learning with graph convolutional networks via importance sampling. In *International Conference on Learning Representations*, 2018.

[13] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 1725–1735. PMLR, 07 2020.

[14] Jacob Cohen. *Statistical Power Analysis for the Behavioral Sciencesstical power analysis for the behavioral sciences*. Lawrence Erlbaum Associates, Hillsdale, NJ, 2 edition, 1988.

[15] Colin Cooper, Martin Dyer, Catherine Greenhill, and Andrew Handley. The flip markov chain for connected regular graphs. *Discrete Applied Mathematics*, 254:56–79, 2019.

[16] Andreea Deac, Marc Lackenby, and Petar Veličković. Expander graph propagation. In *NeurIPS 2022 Workshop: New Frontiers in Graph Learning*, 2022.

[17] Asim Kumar Debnath, Rosa L. Lopez de Compadre, Gargi Debnath, Alan J. Shusterman, and Corwin Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, 34(2):786–797, Feb 1991.

[18] Francesco Di Giovanni, Lorenzo Giusti, Federico Barbero, Giulia Luise, Pietro Lio, and Michael M Bronstein. On over-squashing in message passing neural networks: The impact of width, depth, and topology. In *International Conference on Machine Learning*, pp. 7865–7885. PMLR, 2023.

[19] Paul D. Dobson and Andrew J. Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of Molecular Biology*, 330(4):771–783, 2003.

[20] Vijay Prakash Dwivedi, Ladislav Rampášek, Mikhail Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, and Dominique Beaini. Long range graph benchmark. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

[21] Taoran Fang, Zhiqing Xiao, Chunping Wang, Jiarong Xu, Xuan Yang, and Yang Yang. Dropmessage: Unifying random dropping for graph neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(4):4267–4275, Jun. 2023.

[22] Tomas Feder, Adam Guetz, Milena Mihail, and Amin Saberi. 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)*, pp. 69–76, 2006.

[23] Wenzheng Feng, Jie Zhang, Yuxiao Dong, Yu Han, Huanbo Luan, Qian Xu, Qiang Yang, Evgeny Kharlamov, and Jie Tang. Graph random neural networks for semi-supervised learning on graphs. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 22092–22103. Curran Associates, Inc., 2020.

[24] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[25] Rickard Brüel Gabrielsson, Mikhail Yurochkin, and Justin Solomon. Rewiring with positional encodings for graph neural networks. *Transactions on Machine Learning Research*, 2023.

[26] Yarin Gal and Zoubin Ghahramani. Bayesian convolutional neural networks with bernoulli approximate variational inference, 2016.

[27] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In Maria Florina Balcan and Kilian Q. Weinberger (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1050–1059, New York, New York, USA, 06 2016. PMLR.

[28] Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[29] 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*, pp. 1416–1424. ACM, 2018.

[30] 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, pp. 64–76, New York, NY, USA, 2022. Association for Computing Machinery.

[31] 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, pp. 89–98, New York, NY, USA, 1998. Association for Computing Machinery.

[32] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1263–1272. PMLR, 08 2017.

[33] Francesco Di Giovanni, T. Konstantin Rusch, Michael Bronstein, Andreea Deac, Marc Lackenby, Siddhartha Mishra, and Petar Veličković. How does over-squashing affect the power of GNNs? *Transactions on Machine Learning Research*, 2024.

[34] Jhony H. Giraldo, Konstantinos Skianis, Thierry Bouwmans, and Fragkiskos D. Malliaros. 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, pp. 566–576, New York, NY, USA, 2023. Association for Computing Machinery.

[35] Benjamin Gutteridge, Xiaowen Dong, Michael M Bronstein, and Francesco Di Giovanni. DRew: Dynamically rewired message passing with delay. In *International Conference on Machine Learning*, pp. 12252–12267. PMLR, 2023.

[36] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[37] Arman Hasanzadeh, Ehsan Hajiramezanali, Shahin Boluki, Mingyuan Zhou, Nick Duffield, Krishna Narayanan, and Xiaoning Qian. Bayesian graph neural networks with adaptive connection sampling, 2020.

[38] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[39] Larry V. Hedges. Distribution theory for glass's estimator of effect size and related estimators. *Journal of Educational Statistics*, 6(2):107–128, 2025/03/15/ 1981. Full publication date: Summer, 1981.

[40] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

[41] Lei Huang, Jie Qin, Yi Zhou, Fan Zhu, Li Liu, and Ling Shao. Normalization techniques in training dnns: Methodology, analysis and application, 2020.

[42] John J Irwin, Teague Sterling, Michael M Mysinger, Erin S Bolstad, and Ryan G Coleman. ZINC: a free tool to discover chemistry for biology. *J Chem Inf Model*, 52(7):1757–1768, June 2012.

[43] Bo Jiang, Yong Chen, Beibei Wang, Haiyun Xu, and Bin Luo. Dropagg: Robust graph neural networks via drop aggregation. *Neural Networks*, 163:65–74, 2023.

[44] 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.

[45] Kenji Kawaguchi. Deep learning without poor local minima. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[46] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, San Diega, CA, USA, 2015.

[47] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[48] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, R. Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. In D. Touretzky (ed.), *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1989.

[49] Guohao Li, Matthias Müller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9266–9275, 2019.

[50] Qimai Li, Zhichao Han, and Xiao-ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), 04 2018.

[51] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. Gated graph sequence neural networks. In Yoshua Bengio and Yann LeCun (eds.), *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[52] Derek Lim, Xiuyu Li, Felix Hohne, and Ser-Nam Lim. New benchmarks for learning on non-homophilous graphs. *arXiv preprint arXiv:2104.01404*, 2021.

[53] 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.

[54] Yang Liu, Chuan Zhou, Shirui Pan, Jia Wu, Zhao Li, Hongyang Chen, and Peng Zhang. 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, pp. 221–230, New York, NY, USA, 2023. Association for Computing Machinery.

[55] L. Lovász. Random walks on graphs: A survey. *Combinatorics, Paul Erdos is Eighty*, 2(1): 1–46, 1993.

[56] 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, pp. 155–164, New York, NY, USA, 2005. Association for Computing Machinery.

[57] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163, 07 2000.

[58] Péter Mernyei and Cătălina Cangea. Wiki-cs: A wikipedia-based benchmark for graph neural networks. *arXiv preprint arXiv:2007.02901*, 2020.

[59] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017.

[60] Federico Monti, Michael Bronstein, and Xavier Bresson. Geometric matrix completion with recurrent multi-graph neural networks. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[61] Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020.

[62] Galileo Mark Namata, Ben London, Lise Getoor, and Bert Huang. Query-driven active surveying for collective classification. In *International Workshop on Mining and Learning with Graphs*, Edinburgh, Scotland, 2012. MLG.

[63] Khang Nguyen, Hieu Nong, Vinh Nguyen, Nhat Ho, Stanley Osher, and Tan Nguyen. Revisiting over-smoothing and over-squashing using ollivier-ricci curvature. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

[64] Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *International Conference on Learning Representations*, 2020.

[65] Pál András Papp, Karolis Martinkus, Lukas Faber, and Roger Wattenhofer. Dropgnn: Random dropouts increase the expressiveness of graph neural networks. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 21997–22009. Curran Associates, Inc., 2021.

[66] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. In *International Conference on Learning Representations*, 2020.

[67] Chendi Qian, Andrei Manolache, Kareem Ahmed, Zhe Zeng, Guy Van den Broeck, Mathias Niepert, and Christopher Morris. Probabilistically rewired message-passing neural networks. In *The Twelfth International Conference on Learning Representations*, 2024.

[68] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2020.

[69] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-Scale Attributed Node Embedding. *Journal of Complex Networks*, 9(2), 2021.

[70] T. Konstantin Rusch, Michael M. Bronstein, and Siddhartha Mishra. A survey on oversmoothing in graph neural networks, 2023.

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

[72] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks, 2017.

[73] Ida Schomburg, Antje Chang, Christian Ebeling, Marion Gremse, Christian Heldt, Gregor Huhn, and Dietmar Schomburg. BRENDA, the enzyme database: updates and major new developments. *Nucleic Acids Res*, 32(Database issue):D431–3, January 2004.

[74] S. S. Shapiro and M. B. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591–611, 2025/03/12/ 1965. Full publication date: Dec., 1965.

[75] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *Relational Representation Learning Workshop, NeurIPS 2018*, 2018.

[76] Alistair Sinclair and Mark Jerrum. Approximate counting, uniform generation and rapidly mixing markov chains. *Information and Computation*, 82(1):93–133, 1989.

[77] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[78] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, 2015.

[79] Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. In *International Conference on Learning Representations*, 2022.

[80] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[81] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[82] Nikil Wale and George Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. In *Sixth International Conference on Data Mining (ICDM'06)*, pp. 678–689, 2006.

[83] Zhanghao Wu, Paras Jain, Matthew Wright, Azalia Mirhoseini, Joseph E Gonzalez, and Ion Stoica. Representing long-range context for graph neural networks with global attention. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[84] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5453–5462. PMLR, 07 2018.

[85] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks?, 2019.

[86] Han Xuanyuan, Tianxiang Zhao, and Dongsheng Luo. Shedding light on random dropping and oversmoothing. In *NeurIPS 2023 Workshop: New Frontiers in Graph Learning*, 2023.

[87] Pinar Yanardag and S.V.N. Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pp. 1365–1374, New York, NY, USA, 2015. Association for Computing Machinery.

[88] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. 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, pp. 974–983, New York, NY, USA, 2018. Association for Computing Machinery.

[89] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 5812–5823. Curran Associates, Inc., 2020.

[90] Yuning You, Tianlong Chen, Zhangyang Wang, and Yang Shen. When does self-supervision help graph convolutional networks? In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 10871–10880. PMLR, 07 2020.

[91] Yuning You, Tianlong Chen, Zhangyang Wang, and Yang Shen. L2-gcn: Layer-wise and learned efficient training of graph convolutional networks. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2124–2132, 2020.

[92] Yuning You, Tianlong Chen, Zhangyang Wang, and Yang Shen. 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, pp. 1300–1309, New York, NY, USA, 2022. Association for Computing Machinery.

[93] Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnns. In *International Conference on Learning Representations*, 2020.

[94] Wenqing Zheng, Edward W Huang, Nikhil Rao, Sumeet Katariya, Zhangyang Wang, and Karthik Subbian. Cold brew: Distilling graph node representations with incomplete or missing neighborhoods. In *International Conference on Learning Representations*, 2022.

[95] Kaixiong Zhou, Xiao Huang, Yuening Li, Daochen Zha, Rui Chen, and Xia Hu. Towards deeper graph neural networks with differentiable group normalization. In *Advances in neural information processing systems*, 2020.

[96] Kaixiong Zhou, Xiao Huang, Daochen Zha, Rui Chen, Li Li, Soo-Hyun Choi, and Xia Hu. Dirichlet energy constrained learning for deep graph neural networks. *Advances in neural information processing systems*, 2021.

[97] Kuangqi Zhou, Yanfei Dong, Kaixin Wang, Wee Sun Lee, Bryan Hooi, Huan Xu, and Jiashi Feng. Understanding and resolving performance degradation in deep graph convolutional networks. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 2728–2737, 2021.

[98] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 7793–7804. Curran Associates, Inc., 2020.

[99] Marinka Zitnik and Jure Leskovec. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, 33(14):i190–i198, 07 2017.

Appendix

Table of Contents

A Related Works

A.1 Methods for Alleviating Over-smoothing

A popular choice for reducing over-smoothing in GNNs is to regularize the model. Recall that DropEdge [68] implicitly regularizes the model by adding noise to it (Section 2.2). A similarly regularization effect is observed with the methods discussed in the main text – DropNode [23], DropAgg [43], DropGNN [65], Dropout [77] and DropMessage [21]. Graph Drop Connect (GDC) [37] combines DropEdge and DropMessage together, resulting in a layer-wise sampling scheme that uses a different subgraph for message-aggregation over each feature dimension. These methods successfully addressed the over-smoothing problem, enabling the training of deep GNNs, and performed competitively on several benchmarking datasets.

Table 3: Statistics of node-classification datasets. Homophily measures as defined in [52].

Dataset	Nodes	Edges	Features	Classes	Homophily
Homophilic Networks					
Reddit	232,965	114,615,892	602	41	0.653
OGBN-ArXiv	169,343	1,166,243	128	40	0.416
Coauthor-CS	18,333	163,788	6,805	15	0.755
Coauthor-Physics	34,493	495,924	8,415	5	0.847
Wiki-CS	11,701	216,123	300	10	0.568
Amazon-Computers	13,752	491,722	767	10	0.700
Amazon-Photo	7,650	238,162	745	8	0.772
Heterophilic Networks					
Flickr	89,250	899,756	500	7	0.070
Cornell	183	298	1,703	5	0.031
Texas	183	325	1,703	5	0.001
Wisconsin	251	515	1,703	5	0.094

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 [41]. Exploiting the inductive bias in graph-structured data, normalization techniques like PairNorm [93], Differentiable Group Normalization (DGN) [95] and NodeNorm [97] have been proposed to reduce over-smoothing in GNNs. On the other hand, Energetic Graph Neural Networks (EGNNs) [96] 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) [38] in computer vision, [49] proposed the use of residual connections to prevent the smoothing of representations. Residual connections successfully improved the performance of GCN on a range of graph-learning tasks. [13] introduced GCN-II, which uses skip connections from the input to all hidden layers. This layer wise propagation rule has allowed for training of ultra-deep networks – up to 64 layers. Some other architectures, like the Jumping Knowledge Network (JKNet) [84] and the Deep Adaptive GNN (DAGNN) [53], aggregate the representations from *all* layers, $\{\boldsymbol{z}_i^{(\ell)}\}_{\ell=1}^{L}$, before processing them through a readout layer.

A.2 Homophily Bias in Evaluation of Techniques for Deep GNN

We examine the evaluation protocols commonly used for assessing methods aimed at alleviating over-smoothing in deep GNNs – many of which are also widely adopted for training deep architectures. Notably, we highlight a misalignment between the intended goal of these methods – to improve the trainability of deep GNNs – and their evaluation, which is often restricted to short-range tasks.

For example, DropEdge [68] was evaluated on Cora [57], CiteSeer [31], PubMed [62], and a version of Reddit [36] distinct from the one used in our experiments. The first three exhibit high label homophily (see Table 4) and are known to be better modelled by shallower networks [95]. Reddit also displays strong homophily, as can be seen in Table 3. Similarly, DropNode [23] was evaluated on Cora, CiteSeer, and PubMed; DropAgg [43] on Cora ML, CiteSeer, and OGBN-ArXiv [40], which has moderate homophily; DropMessage [68] was evaluated on Cora, CiteSeer, PubMed, OGBN-ArXiv, and Flickr, with only the latter having low homophily; GDC [37] was evaluated on Cora, Cora ML and CiteSeer.

A similar trend can be observed in the evaluation of feature normalization techniques used to regularize GNNs. PairNorm [93] and DGN [95] were evaluated on Cora, CiteSeer, PubMed, and Coauthor-CS [75]; NodeNorm [97] on Cora, CiteSeer, PubMed, Coauthor-CS, Wiki-CS [58], and Amazon-Photo [75]; and EGNNs [96] on Cora, PubMed, Coauthor-Physics [75], and OGBN-ArXiv – all of these datasets are highly homophilic.

A similar trend is observed in the evaluation of architectural modifications designed to enable deeper GNNs. GCN-II [13] on Cora, CiteSeer, PubMed, and Chameleon; JKNet [84] on Cora, CiteSeer, and Reddit; and DAGNN [53] on Cora, CiteSeer, PubMed, Coauthor-CS, Coauthor-Physics, Amazon-Computers [75], and Amazon-Photo – many of these datasets are highly homophilic as well.

This pattern indicates that an overwhelming proportion of evaluations have been restricted to short-range, homophilic tasks. Such a narrow focus risks overstating the general effectiveness of these methods and masking their potential limitations in long-range scenarios.

A few exceptions stand out. DropGNN [65], which was evaluated on graph-classification from the TUDataset [61], aligning more closely with evaluations of rewiring methods targeting over-squashing [8, 44]. NodeNorm, while primarily evaluated on homophilic datasets, was also tested on three heterophilic graphs: Cornell, Texas, and Wisconsin [66]. GCN-II saw broader evaluation, including on several long-range tasks such as Chameleon, Cornell, Texas, Wisconsin, and the Protein-Protein Interaction (PPI) networks [36]. Lastly, JKNet was also evaluated on the PPI networks.

A.3 Methods for Alleviating Over-squashing

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 a commonality among these methods – edge addition is necessary. As a reminder, *graph rewiring* refers to modifying the edge set of a graph by adding and/or removing edges in a systematic manner. In a special case, which includes many of the rewiring techniques we will discuss, the original topology is completely 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, [4] 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. [79] proposed a curvature-based rewiring strategy, called the Stochastic Discrete Ricci Flow (SDRF), which 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. [8] proposed the Greedy Total Resistance (GTR) technique, which optimizes the graph's total resistance by greedily adding edges to achieve the greatest improvement. 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, [35] 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 entire ℓ-hop receptive field instead of just the immediate neighbors. This results in an improved communication over long distances while also retaining the inductive bias of the shortest distance between nodes.

Spectral methods, on the other hand, use the spectral properties of the matrices encoding the graph topology, e.g. the adjacency or the Laplacian matrix, to design rewiring algorithms. For example, [5] proposed a differentiable graph rewiring layer based on the Lovász bound [55, Corollary 3.3]. Similarly, [6] introduced the Random Local Edge Flip (RLEF) algorithm, which draws inspiration from the "Flip Markov Chain" [22, 56] – 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 [1, 15, 22, 30, 56], 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) [67] learns to probabilistically rewire a graph, effectively mitigating under-reaching as well as over-squashing. Finally, [25] 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.

A.4 Towards a Unified Treatment

Several studies have shown that an inevitable trade-off exists between the problems of over-smoothing and over-squashing, meaning that optimizing for one will compromise the other. For instance, [63, 79] showed that negatively curved edges create bottlenecks in the graph resulting in over-squashing of information. On the other hand, [63, Proposition 4.3] showed that positively curved edges in a graph contribute towards the over-smoothing problem. To address this trade-off, they proposed Batch Ollivier-Ricci Flow (BORF), which adds new edges adjacent to the negatively curved ones, and simultaneously removes positively curved ones. In a similar vein, [34] demonstrated that the minimum number of message-passing steps required to reach a given level of over-smoothing is inversely related to the Cheeger's constant, $h_{\mathcal{G}}$. This again implies an inverse relationship between over-smoothing and over-squashing. To effectively alleviate the two issues together, they proposed the Stochastic Jost and Liu Curvature Rewiring (SJLR) algorithm, which adds edges that 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 [44], 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 the symmetric Laplacian matrix ($\boldsymbol{L}^{\text{sym}} = \boldsymbol{I}_N - (\boldsymbol{D}^{\dagger})^{1/2}\boldsymbol{A}(\boldsymbol{D}^{\dagger})^{1/2}$), and adds edges with the aim of maximizing it. Since the spectral gap directly relates to Cheeger's constant – a measure of bottleneck-edness in the graph – through Cheeger's inequality [2, 3, 76], this directly decreases the over-squashing levels. Moreover, [44, 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, suggesting the method's potential to simultaneously tackle over-smoothing. Taking a somewhat opposite approach, [54] adapted DropEdge to *remove* negatively curved edges sampled from a distribution proportional to edge curvatures. Their method, called CurvDrop, directly reduces over-squashing and, as a side benefit of operating on a sparser subgraph, also mitigates over-smoothing.

B Proofs

B.1 Expected Propagation Matrix under DropEdge-variants

Lemma. *When using DropEdge, the expected propagation matrix is given as:*

$$\mathbb{E}_{\mathsf{DE}}\left[\hat{\boldsymbol{A}}^{(1)}_{ii}\right] = \frac{1-q^{d_i+1}}{(1-q)\,(d_i+1)}$$
$$\mathbb{E}_{\mathsf{DE}}\left[\hat{\boldsymbol{A}}^{(1)}_{ij}\right] = \frac{1}{d_i}\left(1 - \frac{1-q^{d_i+1}}{(1-q)\,(d_i+1)}\right)$$

where $(j \to i) \in \mathcal{E}$*;* $\dot{\boldsymbol{P}}_{ij} = 0$ *otherwise.*

Proof. Recall that under DropEdge, a self-loop is added to the graph *after* the edges are dropped, and then the normalization is performed. In other words, the self-loop is never dropped. Therefore, given the i.i.d. masks, $m_1, \ldots, m_{d_i} \sim \text{Bern}\,(1-q)$, on incoming edges to node i, the total number of messages is given by

$$1 + \sum_{k=1}^{d_i} m_k = 1 + M_i \tag{B.1}$$

where $M_i \sim \text{Binom}\,(d_i, 1-q)$. Under asymmetric normalization (see Section 2.1), the expected weight of the message along the self-loop is computed as follows:

$$\mathbb{E}_{\mathsf{DE}}\left[\hat{\boldsymbol{A}}^{(1)}_{ii}\right] = \mathbb{E}_{m_1,\ldots,m_{d_i}}\left[\frac{1}{1+\sum_{k=1}^{d_i} m_k}\right] \tag{B.2}$$

$$= \mathbb{E}_{M_i}\left[\frac{1}{1+M_i}\right] \tag{B.3}$$

$$= \sum_{k=0}^{d_i}\binom{d_i}{k}(1-q)^k\,(q)^{d_i-k}\left(\frac{1}{1+k}\right) \tag{B.4}$$

$$= \frac{1}{(1-q)\,(d_i+1)}\sum_{k=0}^{d_i}\binom{d_i+1}{k+1}(1-q)^{k+1}\,(q)^{d_i-k} \tag{B.5}$$

$$= \frac{1}{(1-q)\,(d_i+1)}\sum_{k=1}^{d_i+1}\binom{d_i+1}{k}(1-q)^{k}\,(q)^{d_i+1-k} \tag{B.6}$$

$$= \frac{1-q^{d_i+1}}{(1-q)\,(d_i+1)} \tag{B.7}$$

Similarly, if the Bernoulli mask corresponding to $j \to i$ is 1, then the total number of incoming messages to node i is given by

$$2 + \sum_{k=1}^{d_i-1} m_k$$

including one self-loop, which is never dropped, as noted earlier. On the other hand, the weight of the edge is simply 0 if the corresponding Bernoulli mask is 0. Using the Law of Total Expectation, the expected weight of the edge $j \to i$ can be computed as follows:

$$\mathbb{E}_{\mathsf{DE}}\left[\hat{\boldsymbol{A}}^{(1)}_{ij}\right] = q\cdot 0 + (1-q)\,\mathbb{E}_{m_1,\ldots,m_{d_i-1}}\left[\frac{1}{2+\sum_{k=1}^{d_i-1} m_k}\right] \tag{B.8}$$

$$= (1-q)\sum_{k=0}^{d_i-1}\binom{d_i-1}{k}(1-q)^k\,(q)^{d_i-1-k}\left(\frac{1}{2+k}\right) \tag{B.9}$$

$$= \sum_{k=0}^{d_i-1}\frac{(d_i-1)!}{(k+2)!\,(d_i-1-k)!}(1-q)^{k+1}\,(q)^{d_i-1-k}\,(k+1) \tag{B.10}$$

$$= \sum_{k=2}^{d_i+1}\frac{(d_i-1)!}{(k)!\,(d_i+1-k)!}(1-q)^{k-1}\,(q)^{d_i+1-k}\,(k-1) \tag{B.11}$$

$$= \frac{1}{d_i\,(d_i+1)\,(1-q)}\sum_{k=2}^{d_i+1}\binom{d_i+1}{k}(1-q)^{k}\,(q)^{d_i+1-k}\,(k-1) \tag{B.12}$$

$$= \frac{1}{d_i\,(d_i+1)\,(1-q)}\left[(d_i+1)\,(1-q) - 1 + q^{d_i+1}\right] \tag{B.13}$$

$$= \frac{1}{d_i}\left(1-\mathbb{E}_{\mathsf{DE}}\left[\hat{\boldsymbol{A}}^{(1)}_{ii}\right]\right) \tag{B.14}$$

□

Analysis of DropEdge-variants. We will similarly derive the expected propagation matrix for other random edge-dropping algorithms. First off, DropNode [23] samples nodes and drops corresponding columns from the aggregation matrix directly, followed by rescaling of its entries:

$$\mathbb{E}_{\mathsf{DN}}\left[\frac{1}{1-q}\hat{\boldsymbol{A}}\right] = \frac{1}{1-q}\times(1-q)\,\hat{\boldsymbol{A}} = \hat{\boldsymbol{A}} \tag{B.15}$$

That is, the expected propagation matrix is the same as in a NoDrop model ($q = 0$).

Nodes sampled by DropAgg [43] don't aggregate messages. Therefore, if $\hat{\boldsymbol{A}} = \hat{\boldsymbol{A}}^{\text{asym}}$, then the expected propagation matrix is given by

$$\begin{aligned}\mathbb{E}_{\mathsf{DA}}\left[\hat{\boldsymbol{A}}_{ii}\right] &= q + \frac{1-q}{d_i+1} = \frac{1+d_i q}{d_i+1} > \mathbb{E}_{\mathsf{DE}}\left[\hat{\boldsymbol{A}}_{ii}\right] \\ \mathbb{E}_{\mathsf{DA}}\left[\hat{\boldsymbol{A}}_{ij}\right] &= \frac{1}{d_i}\left(1 - \mathbb{E}_{\mathsf{DA}}\left[\hat{\boldsymbol{A}}_{ii}\right]\right) < \mathbb{E}_{\mathsf{DE}}\left[\hat{\boldsymbol{A}}_{ij}\right]\end{aligned} \tag{B.16}$$

Finally, DropGNN [65] samples nodes which neither propagate nor aggregate messages. From any node's perspective, if it is not sampled, then its aggregation weights are computed as for DropEdge:

$$\begin{aligned}\mathbb{E}_{\mathsf{DG}}\left[\hat{\boldsymbol{A}}_{ii}\right] &= q + (1-q)\,\mathbb{E}_{\mathsf{DE}}\left[\hat{\boldsymbol{A}}_{ii}\right] = q + \frac{1-q^{d_i+1}}{d_i+1} > \mathbb{E}_{\mathsf{DA}}\left[\hat{\boldsymbol{A}}_{ii}\right] \\ \mathbb{E}_{\mathsf{DG}}\left[\hat{\boldsymbol{A}}_{ij}\right] &= \frac{1}{d_i}\left(1 - \mathbb{E}_{\mathsf{DG}}\left[\hat{\boldsymbol{A}}_{ii}\right]\right) < \mathbb{E}_{\mathsf{DA}}\left[\hat{\boldsymbol{A}}_{ij}\right]\end{aligned} \tag{B.17}$$

B.2 Sensitivity in L-layer Linear GCNs

Theorem. *In an L-layer linear GCN with $\hat{\boldsymbol{A}} = \hat{\boldsymbol{A}}^{asym}$, using DropEdge, DropAgg or DropGNN decreases the sensitivity of a node $i \in \mathcal{V}$ to another node $j \in \mathbb{S}^{(L)}(i)$, thereby reducing its effective receptive field.*

$$\mathbb{E}_{\dots}\left[\left(\hat{\boldsymbol{A}}^L\right)_{ij}\right] = \sum_{(u_0,\dots,u_L)\in\mathsf{Paths}(j\to i)}\prod_{\ell=1}^{L}\mathbb{E}_{\dots}\left[\hat{\boldsymbol{A}}_{u_\ell u_{\ell-1}}\right] < \mathbb{E}_{\mathsf{ND}}\left[\left(\hat{\boldsymbol{A}}^L\right)_{ij}\right] \tag{B.18}$$

where ND *refers to a NoDrop model ($q = 0$), the placeholder $\cdots$ can be replaced with one of the edge-dropping methods* DE, DA *or* DG*, and the corresponding entries of $\mathbb{E}_{\dots}[\hat{\boldsymbol{A}}]$ can be plugged in from Equation 3.3, Equation B.16 and Equation B.17, respectively. Moreover, the sensitivity monotonically decreases as the dropping probability is increased.*

Proof. Recall that $\dot{\boldsymbol{P}}$ can be viewed as the transition matrix of a non-uniform random walk, such that $\dot{\boldsymbol{P}}_{uv} = \mathbb{P}(u \to v)$. 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 i to j in exactly L hops.

More formally, denote the set of paths connecting the two nodes by

$$\mathsf{Paths}(j \to i) = \{(u_0,\dots,u_L) : u_0 = j; u_L = i; (u_{\ell-1} \to u_\ell) \in \mathcal{E}, \forall \ell \in [L]\} \tag{B.19}$$

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

$$\left(\dot{\boldsymbol{P}}^L\right)_{ij} = \sum_{(u_0,\dots,u_L)\in\mathsf{Paths}(j\to i)}\prod_{\ell=1}^{L}\dot{\boldsymbol{P}}_{u_\ell u_{\ell-1}} \tag{B.20}$$

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

$$\left(\dot{\boldsymbol{P}}^L\right)_{ij} = \sum_{(u_0,\dots,u_L)\in\mathsf{Paths}(j\to i)}\prod_{\ell=1}^{L}\frac{1}{d_{u_\ell}}\left(1 - \frac{1-q^{d_{u_\ell}+1}}{(1-q)\,(d_{u_\ell}+1)}\right) \tag{B.21}$$

$$< \sum_{(u_0,\dots,u_L)\in\mathsf{Paths}(j\to i)}\prod_{\ell=1}^{L}\left(\frac{1}{d_{u_\ell}+1}\right) \tag{B.22}$$

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. From Equation B.16 and Equation B.17, we know that Equation B.22 is true for

DropAgg and DropGNN as well. We conclude the first part of the proof using Equation 3.2 – the sensitivity of node i to node j is proportional to $(\dot{\boldsymbol{P}}^L)_{ij}$.

Next, we recall the geometric series for any q:

$$1 + q + \ldots + q^d = \frac{1 - q^{d+1}}{1 - q} \tag{B.23}$$

Each of the terms on the right are increasing in q, hence, all the $\dot{\boldsymbol{P}}_{u_\ell u_{\ell-1}}$ factors are decreasing in q. Similarly, $\mathbb{E}_{\mathsf{DA}}[\hat{\boldsymbol{A}}_{ij}]$ and $\mathbb{E}_{\mathsf{DG}}[\hat{\boldsymbol{A}}_{ij}]$ decrease with increasing q. Using these results with Equation B.20, we conclude the second part of the theorem.

□

C Theoretical Extensions

C.1 Sensitivity in Nonlinear MPNNs

While linear networks are useful in simplifying the theoretical analysis, they are often not practical. In this subsection, we will consider the upper bounds on sensitivity established in previous works, and extend them to the DropEdge setting.

ReLU GCNs. [84] considered the case of ReLU nonlinearity, so that the update rule is $\boldsymbol{Z}^{(\ell)} = \mathsf{ReLU}(\hat{\boldsymbol{A}}\boldsymbol{Z}^{(\ell-1)}\boldsymbol{W}^{(\ell)})$. Additionally, it makes the simplifying assumption that each path in the computational graph is *active* with a fixed probability, ρ [45, Assumption A1p-m]. Accordingly, the sensitivity (in expectation) between any two nodes is given as

$$\left\| \mathbb{E}_{\mathsf{ReLU}} \left[\frac{\partial \boldsymbol{z}_i^{(L)}}{\partial \boldsymbol{x}_j} \right] \right\|_1 = \left[\rho \left\| \prod_{\ell=1}^{L} \boldsymbol{W}^{(\ell)} \right\|_1 \right] \left(\hat{\boldsymbol{A}}^L \right)_{ij} = \zeta_1^{(L)} \left(\hat{\boldsymbol{A}}^L \right)_{ij} \tag{C.1}$$

where $\zeta_1^{(L)}$ depends only on the depth L, and is independent of the choice of nodes $i, j \in \mathcal{V}$. Taking an expectation w.r.t. the random edge masks, we get

$$\mathbb{E}_{\mathbf{M}^{(1)},\ldots,\mathbf{M}^{(L)}} \left[\left\| \mathbb{E}_{\mathsf{ReLU}} \left[\frac{\partial \boldsymbol{z}_i^{(L)}}{\partial \boldsymbol{x}_j} \right] \right\|_1 \right] = \zeta_1^{(L)} \left(\mathbb{E} \left[\prod_{\ell=1}^{L} \hat{\boldsymbol{A}}^{(\ell)} \right] \right)_{ij} = \zeta_1^{(L)} \left(\mathbb{E} \left[\hat{\boldsymbol{A}} \right]^L \right)_{ij} \tag{C.2}$$

Using Theorem 3.1, we conclude that in a ReLU-GCN, DropEdge, DropAgg and DropGNN will reduce the expected sensitivity between nodes L-hops away. Empirical observations in Figures 1a and 4 suggest that we may expect an increase in sensitivity to neighboring nodes, but a significant decrease in sensitivity to those farther away.

Source-only Message Functions. [8, Lemma 3.2] considers MPNNs with aggregation functions of the form

$$\mathsf{Agg}^{(\ell)} \left(\boldsymbol{z}_i^{(\ell-1)}, \left\{ \boldsymbol{z}_j^{(\ell-1)} : j \in \mathbb{S}^{(1)}(i) \right\} \right) = \sum_{j \in \mathbb{B}^{(1)}(i)} \hat{\boldsymbol{A}}_{ij} \mathsf{Msg}^{(\ell)} \left(\boldsymbol{z}_j^{(\ell-1)} \right) \tag{C.3}$$

and Upd and Msg functions with bounded gradients. In this case, the sensitivity between two nodes $i, j \in \mathcal{V}$ can be bounded as

$$\left\| \frac{\partial \boldsymbol{z}_i^{(L)}}{\partial \boldsymbol{x}_j} \right\|_1 \leq \zeta_2^{(L)} \left(\sum_{\ell=0}^{L} \hat{\boldsymbol{A}}^\ell \right)_{ij} \tag{C.4}$$

As before, we can use the independence of edge masks to get an upper bound on the expected sensitivity:

$$\mathbb{E}_{\mathbf{M}^{(1)},\ldots,\mathbf{M}^{(L)}} \left[\left\| \frac{\partial \boldsymbol{z}_i^{(L)}}{\partial \boldsymbol{x}_j} \right\|_1 \right] \leq \zeta_2^{(L)} \left(\mathbb{E} \left[I_N + \sum_{\ell=1}^{L} \prod_{k=1}^{\ell} \hat{\boldsymbol{A}}^{(k)} \right] \right)_{ij} = \zeta_2^{(L)} \left(\sum_{\ell=0}^{L} \mathbb{E} \left[\hat{\boldsymbol{A}} \right]^\ell \right)_{ij} \tag{C.5}$$

Figure 5 shows the plot of the entries of $\sum_{\ell=0}^{6} \dot{\boldsymbol{P}}^{\ell}$ (i.e. for DropEdge), as in the upper bound above, with $\hat{\boldsymbol{A}} = \hat{\boldsymbol{A}}^{\text{asym}}$. We observe that the sensitivity between nearby nodes marginally increases, while that between distant nodes notably decreases (similar to Figure 1a), suggesting significant over-squashing. Similar observations can be made with $\hat{\boldsymbol{A}} = \hat{\boldsymbol{A}}^{\text{sym}}$, and for other DropEdge-variants.

Source-and-Target Message Functions. [79, Lemma 1] showed that if the aggregation function is instead given by

$$\mathsf{Agg}^{(\ell)}\left(\boldsymbol{z}_i^{(\ell-1)}, \left\{\boldsymbol{z}_j^{(\ell-1)} : j \in \mathbb{S}^{(1)}(i)\right\}\right) = \sum_{j \in \mathbb{B}^{(1)}(i)} \hat{\boldsymbol{A}}_{ij} \mathsf{Msg}^{(\ell)}\left(\boldsymbol{z}_i^{(\ell-1)}, \boldsymbol{z}_j^{(\ell-1)}\right) \tag{C.6}$$

then the sensitivity between nodes $i \in \mathcal{V}$ and $j \in \mathbb{S}^{(L)}(i)$ can be bounded as

$$\left\| \frac{\partial \boldsymbol{z}_i^{(L)}}{\partial \boldsymbol{x}_j} \right\|_1 \leq \zeta_3^{(L)} \left(\hat{\boldsymbol{A}}^L\right)_{ij} \tag{C.7}$$

With random edge-dropping, this bound can be adapted as follows:

$$\mathbb{E}_{\mathbf{M}^{(1)},\ldots,\mathbf{M}^{(L)}} \left[\left\| \frac{\partial \boldsymbol{z}_i^{(L)}}{\partial \boldsymbol{x}_j} \right\|_1 \right] \leq \zeta_3^{(L)} \left(\mathbb{E}\left[\hat{\boldsymbol{A}}\right]^L \right)_{ij} \tag{C.8}$$

which is similar to Equation C.2, only with a different proportionality constant, that is anyway independent of the choice of nodes. Here, again, we invoke Theorem 3.1 to conclude that $(\mathbb{E}[\hat{\boldsymbol{A}}]^L)_{ij}$ decreases monotonically with increasing DropEdge probability q. This implies that, in a non-linear MPNN with $\hat{\boldsymbol{A}} = \hat{\boldsymbol{A}}^{\text{asym}}$, DropEdge lowers the sensitivity bound given above. Empirical results in Figure 4 support the same conclusion for $\hat{\boldsymbol{A}} = \hat{\boldsymbol{A}}^{\text{sym}}$.

C.2 Test-time Monte-Carlo Dropout

Up until now, we have focused on the expected sensitivity of the stochastic representations in models using DropEdge-variants. This corresponds to their training-time behavior, wherein the activations are random. At test-time, the standard practice is to turn these methods off by setting $q = 0$. However, this raises the over-smoothing levels back up [86]. Another way of making predictions is to perform multiple stochastic forward passes, as during training, and then averaging the model outputs. This is similar to Monte-Carlo Dropout, which is an efficient way of ensemble averaging in MLPs [27], CNNs [26] and RNNs [28]. In addition to alleviating over-smoothing, this approach also outperforms the standard implementation in practical settings [86]. We can study the effect of random edge-dropping in this setting by examining the sensitivity of the *expected representations*:

$$\left\| \frac{\partial}{\partial \boldsymbol{x}_j} \mathbb{E}\left[\boldsymbol{z}_i^{(L)}\right] \right\|_1 = \left\| \mathbb{E}\left[\frac{\partial \boldsymbol{z}_i^{(L)}}{\partial \boldsymbol{x}_j} \right] \right\|_1 \tag{C.9}$$

In linear models, the order of the two operations – expectation and 1-norm – is irrelevant:

$$\left\| \mathbb{E}\left[\frac{\partial \boldsymbol{z}_i^{(L)}}{\partial \boldsymbol{x}_j} \right] \right\|_1 = \left\| \mathbb{E}\left[\left(\hat{\boldsymbol{A}}^L\right)_{ij} \right] \boldsymbol{W} \right\|_1 = \mathbb{E}\left[\left(\hat{\boldsymbol{A}}^L\right)_{ij} \|\boldsymbol{W}\|_1 \right] = \mathbb{E}\left[\left\| \frac{\partial \boldsymbol{z}_i^{(L)}}{\partial \boldsymbol{x}_j} \right\|_1 \right] \tag{C.10}$$

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

$$\left\| \frac{\partial}{\partial \boldsymbol{x}_j} \mathbb{E}\left[\boldsymbol{z}_i^{(L)}\right] \right\|_1 \leq \mathbb{E}\left[\left\| \frac{\partial \boldsymbol{z}_i^{(L)}}{\partial \boldsymbol{x}_j} \right\|_1 \right] \leq \ldots \tag{C.11}$$

Therefore, the upper bound results in Appendix C.1 trivially extend to the MC-averaged representations. Although tighter bounds may be derived for this setting, we leave that for future works.

D Empirical Sensitivity Analysis

In this section, we present some supplemental figures demonstrating the negative effects of random edge-dropping, particularly focusing on scenarios not covered by the theory. We also elaborate on the setup used for the empirical sensitivity analysis in Section 3.

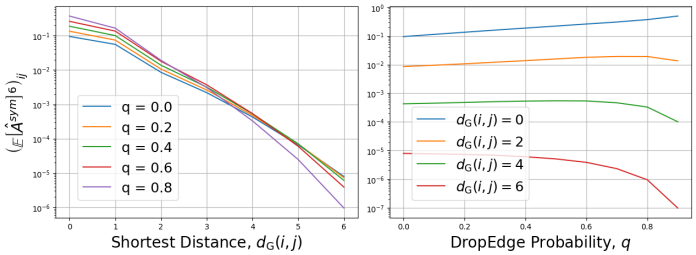


Figure 4: Entries of $\ddot{\boldsymbol{P}}^6$, averaged after binning node-pairs by their shortest distance.

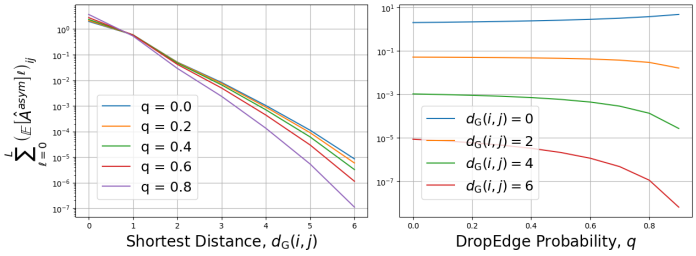


Figure 5: Entries of $\sum_{\ell=0}^{6} \dot{\boldsymbol{P}}^{\ell}$, averaged after binning node-pairs by their shortest distance.

D.1 Symmetrically Normalized Propagation Matrix

The results in Section 3 correspond to the use of $\hat{\boldsymbol{A}} = \hat{\boldsymbol{A}}^{\text{asym}}$ for aggregating messages – in each message passing step, only the in-degree of node i is used to compute the aggregation weights of the incoming messages. In practice, however, it is more common to use the symmetrically normalized propagation matrix, $\hat{\boldsymbol{A}} = \hat{\boldsymbol{A}}^{\text{sym}}$, which ensures that nodes with high out-degree do not dominate the information flow in the graph [47]. As in Equation 3.2, we are looking for

$$\ddot{\boldsymbol{P}}^{L} \coloneqq \mathbb{E}_{\mathbf{M}^{(1)},\dots,\mathbf{M}^{(L)}} \left[\prod_{\ell=1}^{L} \hat{\boldsymbol{A}}^{(\ell)} \right] \tag{D.1}$$

where $\ddot{\boldsymbol{P}} \coloneqq \mathbb{E}_{\mathsf{DE}}[\hat{\boldsymbol{A}}^{\text{sym}}]$. While $\ddot{\boldsymbol{P}}$ is analytically intractable, we can approximate it using Monte-Carlo sampling. Accordingly, we use the Cora dataset, and sample 20 DropEdge masks to compute an approximation of $\ddot{\boldsymbol{P}}$, and plot out the entries of $\ddot{\boldsymbol{P}}^{L}$, as we did for $\dot{\boldsymbol{P}}^{L}$ in Figure 1a. The results are presented in Figure 4, which shows that while the sensitivity between nodes up to 3 hops away is increased, that between nodes farther off is significantly reduced, same as in Figure 1a.

D.2 Upper Bound on Expected Sensitivity

[8] showed that the sensitivity between any two nodes in a graph can be bounded using the sum of the powers of the propagation matrix. In Appendix C.1, we extended this bound to random edge-dropping methods with independent edge masks sampled in each layer:

$$\mathbb{E}_{\mathbf{M}^{(1)},\dots,\mathbf{M}^{(L)}} \left[\left\| \frac{\partial \boldsymbol{z}_i^{(L)}}{\partial \boldsymbol{x}_j} \right\|_1 \right] \leq \zeta_3^{(L)} \left(\sum_{\ell=0}^{L} \mathbb{E} \left[\hat{\boldsymbol{A}} \right]^{\ell} \right)_{ij}$$

Although this bound does not have a closed form, we can again use the Cora network to study its entries. We plot the entries of $\sum_{\ell=0}^{6} \dot{\boldsymbol{P}}^{\ell}$, corresponding to DropEdge, against the shortest distance between node-pairs. The results are presented in Figure 5. We observe an exponential decrease in the sensitivity bound as the distance between nodes increases, suggesting that DropEdge is not suitable for capturing LRIs.

Table 4: Statistics of node-classification datasets. Homophily measures from [52].

Dataset	Nodes	Edges	Features	Classes	Homophily
Homophilic Networks					
Cora	2,708	10,556	1,433	7	0.766
CiteSeer	3,327	9,104	3,703	6	0.627
PubMed	19,717	88,648	500	3	0.664
Heterophilic Networks					
Chameleon	2,277	36,051	2,325	5	0.062
Squirrel	5,201	216,933	2,089	5	0.025
Actor	7,600	29,926	931	5	0.011
TwitchDE	9,498	306,276	128	2	0.142

D.3 MC-Approximation of Sensitivity in Nonlinear MPNNs

Given a target node from the Cora dataset [57], we computed the sensitivity of its representation to source nodes up to $L = 6$ hops away in ReLU-GCNs of width 32. The raw sensitivities were normalized to obtain *influence scores* [84]. This was repeated for 25 target nodes, and 25 model−dropout samples were used for each of them. The source nodes were binned by the shortest distance from the corresponding target node, and the influence scores were averaged over each bin to obtain an *average influence* from nodes ℓ-hops away.

Why influence scores?

E Experiments Details

In this section, we expand on the details of the experiments in Section 5. All experiments were run on a server equipped with an Intel(R) Xeon(R) E5-2620 v3 CPU, 62 GB of RAM, 4 $\times$ NVIDIA GeForce GTX TITAN X GPU (12 GB VRAM each), and CUDA version 12.4.

E.1 Descriptions of the Datasets

Synthetic Datasets. The SyntheticZINC dataset [33], as the name suggests, is a synthetic dataset derived from the ZINC chemical dataset [42], with the dataset size constrained to 12K molecular graphs [20]. Specifically, given a molecular graph $\mathcal{G}$, we set all its nodes' 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. learning 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 the commute time [10] between them is the α^{th} quantile of the distribution of commute times over $\mathcal{G}$. We analyze the effect of underlying mixing on model performance by varying α as $0.1, 0.2, \ldots, 1.0$. The MPNN is chosen to be an L-layer GCN with a MAX-pooling readout, which encourages the model to learn the mixing by effectively passing messages [33, Theorem 3.2]. The model depth is set at $L = \max_{\mathcal{G}} \lceil \text{diam}(\mathcal{G})/2 \rceil = 11$ to ensure that the GCN does not suffer from under-reaching [4, 7].

Node-classification Tasks. Cora [57], CiteSeer [31] and PubMed [62] 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 are represented by a binary vector, where each index indicates whether a specific word from a dictionary is present or absent. Several studies have showed that these datasets have high homophily in node labels [52, 98] and that they are modelled much better by shallower networks than by deeper ones [95]. Chameleon and Squirrel [69] are networks of English Wikipedia web pages on the respective topics, and the edges between web pages indicate links between them. The task is to predict the average-monthly traffic on each of the web pages. The Actor dataset is induced from a larger film-director-actor-writer network [66]. It is a network of film actors, with edges between those that occur on the same Wikipedia page, and node features are binary vectors denoting the presence of specific keywords in the corresponding

Table 5: Statistics of graph-classification datasets.

Dataset	Graphs	Nodes	Edge	Features	Classes
Mutag	188	17.9	39.6	7	2
Proteins	1,113	39.1	145.6	3	2
Enzymes	600	32.6	124.3	3	6
Collab	5,000	74.5	4914.4	0	3
IMDb	1,000	19.8	193.1	0	2
Reddit	2,000	429.6	995.5	0	2

Wikipedia entries. The task is to classify actors into five categories based on the content of their Wikipedia pages. Finally, TwitchDE [69] is a network of Twitch users in Germany, with the edges between them representing their mutual follower relationships. The node features are embeddings of the games played by the users, and the task is to predict whether the users use explicit language.

Graph-classification Tasks. Following [8, 44], we use the graph-classification datasets introduced in [61], which were hypothesized to be long-range tasks. On one hand we have the molecular datasets Mutag, Proteins and Enzymes, and on another we have the social networks Reddit-Binary, IMDb-Binary and Collab. Their statistics are presented in Table 5.

Mutag [17] consists of nitroaromatic compounds, with the objective of predicting their mutagenicity, i.e. the ability to cause genetic mutations, in cells in *Salmonella typhimurium*. Each compound is represented as a graph, where nodes correspond to atoms, represented by their type using one-hot encoding, and edges denote the bonds between them. Proteins [9, 19] is a collection of proteins classified as enzymes or not. The molecules are represented as a graph of amino acids, with edges between those separated up to 6Å apart. The Enzymes [9, 73] dataset is represented similarly, and the task is to classify the proteins into one of 6 Enzyme Commission (EC) numbers – a system to classify enzymes based on the reactions they catalyze.

Collab [87] is a scientific collaboration dataset where each graph represents a researcher's ego network. Nodes correspond to researchers and their collaborators, with edges indicating co-authorship. Each network is labeled based on the researcher's field, which can be High Energy Physics, Condensed Matter Physics, or Astrophysics. IMDb-Binary [87] is a movie collaboration dataset containing the ego-networks of 1,000 actors and actresses from IMDB. In each graph, nodes represent actors, with edges connecting those who have co-starred in the same film. Each graph is derived from either the Action genre or Romance. Finally, Reddit-Binary [87] comprises graphs representing online discussions on Reddit, where nodes correspond to users and edges indicate interactions through comment responses. Each graph is labeled based on whether it originates from a Q&A or discussion-based subreddit.

For the molecular datasets, we use the node features supplied by `PyG` [24], but since they are unavailable for the social networks, we set scalar features $\boldsymbol{x}_i = (1)$ for all nodes in these datasets, following [44].

E.2 Training Configurations

Dataset Splits. For the SyntheticZINC task, we use the train-val-test splits provided in `PyG`. For the homophilic (citation) networks, we use the 'full' split [12], as provided in `PyG`, and for the heterophilic networks, we randomly sample 60% of the nodes for training, 16% for validation, and 24% for testing. On the other hand, for the graph classification tasks, we sample 80% of the graphs for training, and 10% each for validation and testing, following [8, 44].

Model Architecture. We standardize most of the hyperparameters across all experiments to isolate the effect of random dropping. Specifically, we use symmetric normalization of the adjacency matrix to compute the edge weights for GCN, and we set the number of attentions heads for GAT to 2 in order to keep the computational load manageable, while at the same time harnessing the expressiveness of the multi-headed self-attention mechanism. For the SyntheticZINC dataset, we fix the size of the hidden representations at 16, while we fix them to 64 for all the real-world datasets. In all settings, a linear transformation is applied to the node features before message-passing. Afterwards,

a bias term is added and then the ReLU nonlinearity is applied. Finally, a linear readout layer is used to compute the regressand (for regression tasks) or logits (for classification).

Dropping Probability. For the synthetic datasets, we experiment with a NoDrop baseline, and DropEdge, Dropout and DropMessage, each with $q = 0.2$ and $q = 0.5$. For the real-world datasets, the dropping probabilities are varied as $q = 0.1, 0.2, \ldots, 0.9$, so as to reliably find the best performing configuration. We adopt the common practice of turning the dropping methods off at test-time ($q = 0$), isolating the effects on optimization and generalization, which our theory does not address.

DropSens Configurations. For DropSens, we use 4 possible values for proportion of information preserved over corss-edges, $c = 0.5, 0.8, 0.9, 0.95$. Since the dropping probability, q_i, increases with the in-degree of the target node, d_i, the proportion of all edges dropped could become very high, especially with large c. Therefore, we clip the value of q_i by 4 possible choices, $q_i \leq q_{\max} \in \{0.2, 0.3, 0.5, 0.8\}$. We exclude the following configurations: $(c, q_{\max}) \in \{(0.5, 0.2), (0.5, 0.3), (0.5, 0.5), (0.8, 0.2), (0.8, 0.3)\}$, since they use the same dropping probability ($= q_{\max}$) for each edge, and are therefore equivalent to DropEdge. In summary, we test with a total of 11 configurations for DropSens to find the best one for each task.

Optimization Algorithm. The models are trained using the Adam optimizer [46]. On the SyntheticZINC dataset, the models are trained with a learning rate of 2×10^{-3} and a weight decay of 1×10^{-4}, for a total of 200 epochs. On the real-world datasets, we use a learning rate of 1×10^{-3} and no weight decay, following [8, 44]. Here, we cap the maximum number of epochs at 300. In both cases, the learning rate is reduced by a factor of 1×10^{-1} if the validation loss fails to improve beyond a relative threshold of 1×10^{-4} for 10 epochs, again following [8, 44].

Number of Independent Runs. We perform only 10 independent runs on the SyntheticZINC dataset due to its consistently low variance in performance, as also observed by [33]. For real-world datasets, we conduct 20 runs to identify the best-performing dropping configurations. We then perform a one-sided t-test to assess whether dropout improves performance, using a 90% confidence-level ($\alpha = 0.1$) and targeting a statistical power of 0.9 ($\beta = 0.1$). Under the assumption that the dropping method offers superior performance, detecting a medium effect size of 0.5 [14] requires approximately 53 samples per group according to standard power analysis. We round this to 50, and accordingly perform 30 additional runs for the final comparison of the best-performing dropping configuration with the NoDrop baseline.

E.3 DropSens Implementation

```python
import warnings
import sympy
from sympy.abc import x as q
import torch
from torch_geometric.utils import degree, contains_self_loops


def drop_sens(
    edge_index: torch.Tensor,
    c: float,
    max_drop_prob: float = None
):

    if max_drop_prob is None:
        max_drop_prob = 1.

    # Assuming edge index does not have self loops
    if contains_self_loops(edge_index):
        warnings.warn("Degree computation in DropSens assumes absence",
            "of self-loops, but the edge_index passed has them.")
    degrees = degree(edge_index[1]).int()       # Node index -> in-degree

    ds = torch.unique(degrees).tolist()         # Sorted array
    mapper = torch.nan * torch.ones(ds[-1]+1)
    mapper[ds] = max_drop_prob                  # Node degree -> dropping prob

    for d_i in ds:
        q_i = float(sympy.N(sympy.real_roots(
            (1-c)*d_i*(1-q) - q + q**(d_i+1))[-2]   # Following Equation 4.1
        )) if d_i > 0 else 0.
        if q_i > max_drop_prob:
            # Because q monontonic wrt d, and ds is sorted
            break
```

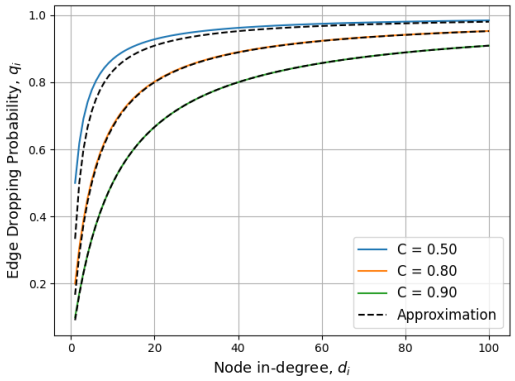


Figure 6: Edge-wise dropping probabilities under DropSens for varying values of c, along with corresponding approximations as in Equation E.1.

```
        mapper[d_i] = q_i

    in_degrees = degrees[edge_index[1]]          # Edge index -> in-degree of target node
    qs = mapper[in_degrees]                      # Edge index -> dropping probability
    edge_mask = qs <= torch.rand(edge_index.size(1))
    edge_index = edge_index[:, edge_mask]

    return edge_index, edge_mask
```

Listing 1: DropSens Implementation

In Listing 1, we present the DropSens implementation used in our experiments, relying mainly on `SymPy` [59].

Unfortunately, computing the roots of Equation 4.1 becomes slow when the in-degree d_i is large – a common scenario in large networks. This issue is especially pronounced when the proportion of information preserved c is large, as the dropping threshold is only met at a higher value of d_i. To address this computational challenge, we propose an approximation:

$$1 - c = \frac{q_i - q_i^{d_i+1}}{d_i\,(1 - q_i)} \approx \frac{q_i}{d_i\,(1 - q_i)} \quad \Longrightarrow \quad q_i \approx \frac{(1-c)\,d_i}{1 + (1-c)\,d_i} \tag{E.1}$$

This approximation becomes increasingly accurate as c increases – since more information needs to be preserved, q_i needs to be small, and hence, $q_i^{d_i+1} \to 0$.

Figure 6 shows the DropSens probabilities for masking incoming messages based on the in-degree of the target nodes, along with the corresponding approximations. It is clear to see that the approximation gets increasingly more accurate for lower proportions of information loss.

F Supplementary Experiments

F.1 Test Accuracy versus DropEdge Probability

In Section 3, we studied the effect of edge-dropping probability on sensitivity between nodes at different distances. However, this analysis may be insufficient to precisely predict the impact on model performance since DropEdge-variants significantly affect the optimization trajectory as well. To learn more about the relationship between test-time performance and dropping probability, we evaluate DropEdge-GCNs on the heterophilic datasets; the results are shown in Figure 7. Clearly, on Chameleon, Squirrel and TwitchDE, the performance degrades with increasing dropping probability, as was suggested by Theorem 3.1 and Figure 1a. Surprisingly, the trends are significantly monotonic with GCNs of all depths, $L = 2, 4, 6, 8$.

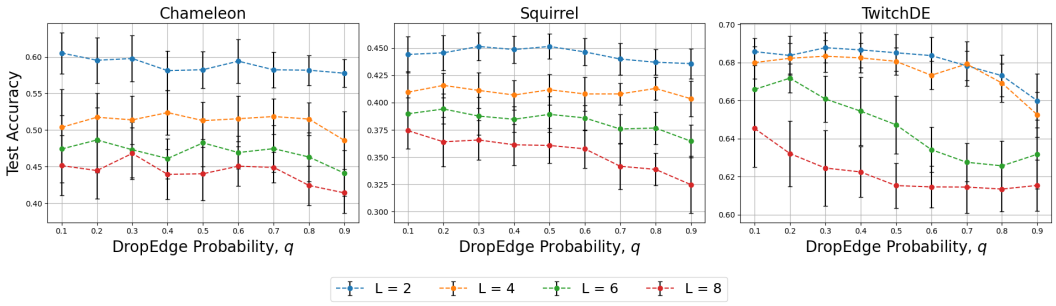


Figure 7: Dropping probability versus test accuracy of DropEdge-GCN. The theory the explains the contrasting trends as follows: random edge-dropping pushes models to fit to local information during training, which is suitable for short-range tasks, but harms test-time performance in long-range ones.

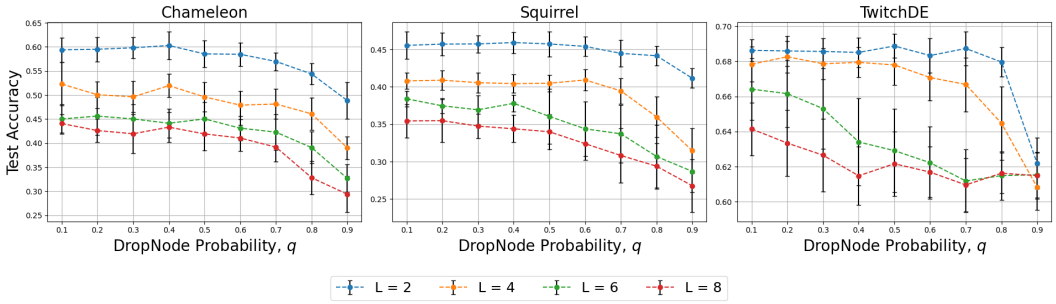


Figure 8: Dropping probability versus test accuracy of DropNode-GCN.

F.2 Remark on DropNode

In Equation B.15, we noted that DropNode does not suffer from loss in sensitivity. However, those results were in expectation. Moreover, our analysis did not account for the effects on the learning trajectory. In practice, a high DropNode probability would make it hard for information in the node features to reach distant nodes. This would prevent the model from learning to effectively combine information from large neighborhoods, harming generalization. In Figure 8, we visualize the relationship between test-time performance and DropNode probability. The performance monotonically decreases with increasing dropping probability, as was observed with DropEdge.

F.3 Over-squashing or Under-fitting?

The results in the previous subsection suggest that using random edge-dropping to regularize model training leads to poor test-time performance. We hypothesize that this occurs because the models struggle to propagate information over long distances, causing node representations to overfit to local neighborhoods. However, a confounding effect is at play: DropEdge variants reduce the generalization gap by preventing overfitting to the training set, i.e. poorer training performance. If this regularization is too strong, it could lead to underfitting, which could also explain the poor test-time performance on heterophilic datasets. This concern is particularly relevant because the heterophilic networks are much larger than homophilic ones (see Table 4), making them more prone to underfitting. To investigate this, we plot the training accuracies of deep DropEdge-GCNs on the heterophilic datasets; Figure 9 shows the results. It is clear that the models do not underfit as the dropping probability increases. In fact, somewhat unexpectedly, the training metrics improve. Together with the results in Figure 7, we conclude that DropEdge-like methods are detrimental in long-range tasks since they cause overfitting to short-range artifacts in the training data, resulting in poor generalization at test-time.

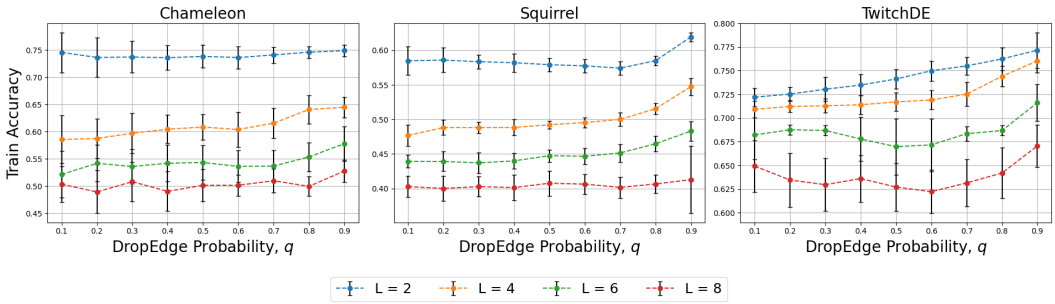


Figure 9: DropEdge probability versus training accuracy of GCNs. The training performance improves with q, suggesting that the models are not underfitting. Instead, the reason for poor test-time performance (Figure 7) is that models are over-fitting to short-range signals during training, resulting in poor generalization.

Table 6: Difference in mean test accuracy (%) between the best performing configuration of each dropout method and the baseline NoDrop model, with GAT as the base model. Cell colors represent p-values from a t-test evaluating whether dropout improves performance.

(a) Node classification tasks.

Dropout	Homophilic Networks			Heterophilic Networks		
	Cora	CiteSeer	PubMed	Chameleon	Squirrel	TwitchDE
DropEdge	+0.483	+0.828	−0.064	−1.988	−1.125	−0.206
DropNode	+0.200	−0.002	−0.181	−6.090	−2.016	−2.383
DropAgg	+0.322	+0.797	+0.032	−7.779	−4.904	−1.014
DropGNN	+0.519	+1.058	−0.149	−10.572	−5.214	−1.636
Dropout	+0.600	+0.104	+0.265	−7.891	−2.773	−1.598
DropMessage	+0.389	+0.039	+0.633	−1.569	−0.092	−0.139

(b) Graph classification tasks.

Dropout	Molecular Networks			Social Networks		
	Mutag	Proteins	Enzymes	Reddit	IMDb	Collab
DropEdge	+0.900	−0.375	+0.290	+0.550	−0.120	0.000
DropNode	+0.400	−0.196	−5.085	+1.860	+3.760	0.000
DropAgg	+1.200	+0.196	+0.519	+0.400	+0.860	0.000
DropGNN	+1.800	+0.143	−3.658	−0.140	+1.220	0.000
Dropout	+1.100	−0.679	−8.313	+1.690	+3.340	0.000
DropMessage	+3.600	−0.107	−3.382	+1.300	+2.680	0.000

G Supplementary Experimental Results

G.1 Performance of GAT with Dropping Methods

In Table 6, we present the results of experiments in Section 5.2, but with the GAT architecture. For node classification tasks, we see the same dichotomy as in Table 1a, with dropping methods significantly improving performance on homophilic networks, while being detrimental to performance on heterophilic networks. On graph classification tasks, the dropping methods improve performance, but the improvement (if any) is not statistically significant (in $28/36 \approx 78\%$ cases). Note that the GAT architecture was unable to learn the Collab dataset, i.e. the performance in all cases was as good as a random classifier's.

G.2 Effect Size in Statistical Tests

The reliance on p-values as a measure of statistical significance has been widely criticized due to its limitations in conveying the magnitude of an effect. Although a low p-value indicates that an observed difference is unlikely to have occurred under the null hypothesis, it does not provide information about the practical significance of the result. A statistically significant effect may be too

Table 7: Hedges' g statistic for different dataset−model−dropout combinations. Color-coding for effect size according to [14]; red denotes negative effect, and green denotes positive effect. Medium to large positive effect sizes in bold.

No effect | Small effect | Medium effect | Large effect

Dataset	GNN	DropEdge	DropNode	DropAgg	DropGNN	Dropout	DropMessage
Cora	GCN	**+0.973**	**+0.759**	**+0.431**	**+0.900**	**+0.990**	+0.060
	GIN	−0.174	+0.205	+0.068	−0.212	**+1.010**	**+2.180**
	GAT	**+0.933**	+0.317	**+0.455**	**+1.083**	**+0.977**	**+0.710**
CiteSeer	GCN	**+0.848**	+0.268	**+0.628**	**+0.929**	+0.155	−0.397
	GIN	−0.254	**+0.515**	−0.124	−0.854	+0.006	+0.132
	GAT	**+0.799**	−0.002	**+0.777**	**+0.985**	+0.103	+0.040
PubMed	GCN	**+1.039**	**+1.285**	−0.627	**+1.455**	**+2.625**	**+2.907**
	GIN	−0.043	+0.343	−0.546	−1.213	**+0.423**	**+1.118**
	GAT	−0.108	−0.320	+0.053	−0.271	**+0.407**	**+0.964**
Chameleon	GCN	−0.167	−0.174	−3.279	−0.479	−0.385	**+0.356**
	GIN	−0.219	−0.567	−0.329	−0.510	−0.857	+0.290
	GAT	−0.429	−1.326	−1.945	−2.438	−1.504	−0.333
Squirrel	GCN	+0.006	−0.389	−8.183	−0.180	−0.066	+0.189
	GIN	+0.179	−0.204	+0.199	−0.052	−0.111	+0.085
	GAT	−0.393	−0.750	−1.899	−2.251	−1.165	−0.038
TwitchDE	GCN	−0.099	−0.113	−2.272	−0.426	−0.280	+0.165
	GIN	−0.079	−0.116	+0.140	−0.005	−0.389	−0.016
	GAT	−0.201	−1.206	−0.726	−0.938	−0.921	−0.129
Mutag	GCN	−0.087	−0.448	−0.069	−0.017	−0.026	+0.174
	GIN	−0.107	−0.351	−0.330	−0.640	−0.240	−0.342
	GAT	+0.071	+0.034	+0.099	+0.151	+0.091	+0.281
Proteins	GCN	+0.339	**+0.450**	+0.260	+0.307	**+0.397**	**+0.477**
	GIN	−0.331	−0.541	−0.352	−0.491	−0.657	−0.217
	GAT	−0.076	−0.042	+0.040	+0.031	−0.136	−0.021
Enzymes	GCN	−0.072	−0.272	−0.055	−0.359	−0.758	−0.543
	GIN	−0.673	−0.070	−0.083	−0.548	−0.290	−0.048
	GAT	+0.031	−0.508	+0.055	−0.380	−0.960	−0.353
Reddit	GCN	−1.712	−1.333	−3.075	−2.360	−1.065	−1.527
	GIN	−0.531	+0.095	+0.163	+0.240	**+0.503**	+0.165
	GAT	+0.209	**+0.638**	+0.154	−0.054	**+0.632**	**+0.469**
IMDb	GCN	+0.246	**+0.513**	**+0.642**	+0.281	+0.200	+0.240
	GIN	−0.364	−0.024	−0.327	−0.932	−0.204	−0.090
	GAT	−0.030	**+0.917**	+0.245	+0.344	**+0.885**	**+0.650**
Collab	GCN	−0.195	−0.859	−6.962	+0.140	−0.348	−0.065
	GIN	−0.220	**+0.396**	−0.916	−1.139	−0.071	**+0.396**
	GAT	+0.000	+0.000	+0.000	+0.000	+0.000	+0.000

small to be meaningful in real-world applications, while a non-significant result does not necessarily imply the absence of a meaningful effect, particularly when sample sizes are small. These concerns have led to an increased emphasis on effect size measures, which quantify the magnitude of differences independently of sample size.

One widely used measure of effect size is Cohen's d statistic [14], which standardizes the difference between two group means by dividing by the pooled standard deviation:

$$d = \frac{\bar{x}_1 - \bar{x}_2}{s_p} \tag{G.1}$$

$$s_p = \sqrt{\frac{(n_1 - 1)\, s_1^2 + (n_2 - 1)\, s_2^2}{n_1 + n_2 - 2}} \tag{G.2}$$

where $\bar{x}_1$ and $\bar{x}_2$ are sample means, s_1^2 and s_2^2 are unbiased sample variance, and n_1 and n_2 are the sizes of the samples. However, Cohen's d assumes that the sample standard deviation is an unbiased estimator of the population standard deviation. In small samples, this assumption does not hold, as the sample standard deviation tends to underestimate the true population variability. To address this

Table 8: Performance of GIN with different rewiring methods in graph-classification tasks, following [8, 44]. **First**, **second**, and **third** best results are colored.

Rewiring	Mutag	Proteins	Enzymes	Reddit	IMDb	Collab
None	77.70	70.80	33.80	86.79	70.18	**72.99**
Last FA	**83.45**	**72.30**	**47.40**	**90.22**	**70.91**	**75.06**
Every FA	72.55	70.38	28.38	50.36	49.16	32.89
DIGL	**79.70**	70.76	**35.72**	76.04	64.39	54.50
SDRF	**78.40**	69.81	**35.82**	86.44	69.72	72.96
FoSR	78.00	**75.11**	29.20	**87.35**	**71.21**	**73.28**
GTR	77.60	**73.13**	30.57	**86.98**	**71.28**	72.93
DropSens	70.60	68.00	29.56	76.44	62.48	65.77

bias, Hedges' g statistic [39] introduces a correction factor that adjusts Cohen's d statistic for small sample sizes:

$$g \approx \left(1 - \frac{3}{4\left(n_1 + n_2\right) - 9}\right) d \tag{G.3}$$

[14] suggested that an effect size of 0.2 be considered small, 0.5 be considered medium, and 0.8 be considered large. In Table 7, we present Hedges' g statistic for the statistical tests in Section 5.2. We can clearly see that for homophilic datasets, there is a strong *positive effect* of using the dropping methods, but for heterophilic datasets and graph classification datasets, there is *at most* a small positive effect of using the dropping methods; rather, in most cases there is a *negative effect*.

G.3 Performance of GIN with DropSens

In Section 5.3, we showed that when modelling long-range graph-classification tasks using GCNs, DropSens outperforms state-of-the-art graph-rewiring techniques designed for alleviating over-squashing. However, it does not perform as well with GIN, as can be seen in Table 8 – unsurprising, since DropSens was specifically designed to work with GCN's message-passing scheme. This highlights the main limitation of DropSens, necessitating architecture-specific alteration to the edge-dropping strategy, which is not practical in general.

G.4 Best-performing Dropping Probabilities

For the real-world datasets in Section 5.2, we report the best performing dropping probability for different dataset−model−dropout combinations in Table 9.

Table 9: Best performing dropout configuration – $q_{\max}$ and c for DropSens, and q for other dropping methods.

GNN	Dataset	DropEdge	DropNode	DropAgg	DropGNN	Dropout	DropMessage	DropSens
Cora	GCN	0.7	0.4	0.1	0.7	0.7	0.1	0.5, 0.95
	GIN	0.1	0.2	0.1	0.1	0.3	0.5	0.2, 0.90
	GAT	0.8	0.3	0.9	0.4	0.7	0.6	0.8, 0.50
CiteSeer	GCN	0.9	0.1	0.9	0.8	0.1	0.3	0.2, 0.95
	GIN	0.1	0.4	0.1	0.1	0.1	0.2	0.3, 0.95
	GAT	0.9	0.2	0.9	0.8	0.1	0.1	0.8, 0.50
PubMed	GCN	0.3	0.4	0.1	0.2	0.5	0.7	0.5, 0.95
	GIN	0.1	0.1	0.1	0.1	0.1	0.6	0.2, 0.90
	GAT	0.1	0.1	0.1	0.1	0.5	0.8	0.3, 0.95
Chameleon	GCN	0.4	0.1	0.1	0.1	0.4	0.5	0.5, 0.80
	GIN	0.1	0.1	0.2	0.1	0.1	0.1	0.2, 0.95
	GAT	0.1	0.1	0.1	0.1	0.2	0.1	0.2, 0.95
Squirrel	GCN	0.2	0.6	0.9	0.3	0.5	0.6	0.5, 0.90
	GIN	0.2	0.5	0.2	0.5	0.3	0.7	0.2, 0.95
	GAT	0.1	0.1	0.1	0.1	0.1	0.1	0.2, 0.95
TwitchDE	GCN	0.3	0.2	0.9	0.1	0.1	0.6	0.3, 0.95
	GIN	0.2	0.3	0.1	0.2	0.1	0.5	0.2, 0.90
	GAT	0.1	0.1	0.1	0.1	0.1	0.1	0.2, 0.90
Actor	GCN	0.9	0.1	0.9	0.5	0.2	0.1	0.2, 0.95
	GIN	0.5	0.1	0.2	0.3	0.6	0.2	0.2, 0.90
	GAT	0.9	0.2	0.8	0.7	0.1	0.1	0.8, 0.50
Mutag	GCN	0.5	0.8	0.7	0.2	0.2	0.3	0.5, 0.80
	GIN	0.1	0.3	0.1	0.6	0.2	0.5	0.5, 0.95
	GAT	0.4	0.1	0.2	0.1	0.4	0.4	0.5, 0.95
Proteins	GCN	0.1	0.1	0.1	0.2	0.1	0.3	0.8, 0.90
	GIN	0.1	0.3	0.1	0.1	0.4	0.1	0.5, 0.95
	GAT	0.4	0.1	0.2	0.1	0.1	0.1	0.8, 0.95
Enzymes	GCN	0.5	0.1	0.1	0.6	0.1	0.4	0.8, 0.80
	GIN	0.3	0.1	0.1	0.1	0.1	0.1	0.8, 0.95
	GAT	0.1	0.1	0.1	0.1	0.1	0.1	0.5, 0.90
Reddit	GCN	0.1	0.1	0.1	0.1	0.2	0.1	0.2, 0.95
	GIN	0.1	0.1	0.1	0.1	0.1	0.1	0.2, 0.95
	GAT	0.9	0.7	0.6	0.7	0.6	0.9	0.8, 0.95
IMDb	GCN	0.6	0.7	0.1	0.6	0.1	0.9	0.8, 0.50
	GIN	0.1	0.3	0.1	0.1	0.2	0.2	0.2, 0.95
	GAT	0.7	0.7	0.1	0.6	0.9	0.6	0.2, 0.95
Collab	GCN	0.1	0.1	0.4	0.1	0.1	0.1	0.2, 0.95
	GIN	0.1	0.2	0.1	0.1	0.2	0.1	0.2, 0.90
	GAT	0.1	0.1	0.1	0.1	0.1	0.1	0.2, 0.90