Half-Hop: A graph upsampling approach for slowing down message passing

Anonymous Authors[1]

Abstract

Message passing neural networks have shown a lot of success on graph-structured data. However, there are many instances where message passing can lead to over-smoothing or fail when neighboring nodes belong to different classes. In this work, we introduce a simple yet general framework for improving learning in message passing neural networks. Our approach essentially upsamples edges in the original graph by adding "slow nodes" at each edge that can mediate communication between a source and a target node. Our method only modifies the input graph, making it plug-and-play and easy to use with existing models. To understand the benefits of slowing down message passing, we provide theoretical and empirical analyses. We report results on several supervised and self-supervised benchmarks, and show improvements across the board, notably in heterophilic conditions where adjacent nodes may have different labels. We also show how our method can be used to generate multi-scale views for graph self-supervised learning.

1. Introduction

Graph neural networks (GNNs) are now a widely used tool, with applications in recommender systems (Ying et al., 2018), drug discovery (Stokes et al., 2020; Gaudelet et al., 2021), and much more (Monti et al., 2017; Cui et al., 2019; Ktena et al., 2017). However, because graphs are highly variable, building general approaches for learning on graphs that work robustly on many different problems has been a major challenge.

Most GNNs rely on message passing (MP) and aggregation to perform inference (Gilmer et al., 2017). Message passing, while intuitive and simple, can also be limiting in some cases (Alon & Yahav, 2021; Topping et al., 2021; Oono & Suzuki, 2020). This is especially true when working with complex and heterophilic graphs where nodes from different classes are connected, and in cases where the degree distributions and connectivity is varied throughout the network (Yan et al., 2021). Finding ways to mitigate these problems is of great importance for advancing GNNs, and to do so, we need flexible and generalizable strategies that can easily be applied to both supervised and unsupervised losses.

In this work, we introduce Half-Hop, a simple yet general framework for improving learning in message passing neural networks. The main idea behind our approach is to upsample the input graph: we do this through the introduction of "slow nodes" along each edge that slows-down the messages sent by the source node to the target node. Rather than making explicit modifications of our loss or encoder, we simply modify the input graph, making it plug-and-play and easy to use with existing models and architectures.

We apply our graph upsampling approach to a wide range of benchmark datasets used in supervised and self-supervised node classification tasks. Across the board, we find that our approach provides improvements to the backbones and baseline models that we tested. In self-supervised learning, we demonstrate impressive boosts in performance when applying our approach to state-of-the-art models for self-supervised learning (SSL) (i.e., GRACE (Zhu et al., 2020b), BGRL (Thakoor et al., 2021)). Overall, these results suggest that Half-Hop can significantly improve the performance of GNNs in a wide range of different types of graphs, across models, losses, and tasks.

The contributions of this work include:

- *A graph upsampling approach that improves node classification for a range of GNNs:* In extensive experiments, we show the utility of graph upsampling in both supervised and unsupervised settings.
- *Novel graphs augmentations for SSL:* An important challenge in using SSL on graphs is the design of augmentations to create different views of a graph for contrastive learning (Zhu et al., 2021). We demonstrate that Half-Hop can be used as a standalone augmentation or in conjunction with other augmentations to improve upon state-of-the-art SSL models.
- *Plug-and-play component to improve message passing on heterophilic graphs:* Our results on heterophilic

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

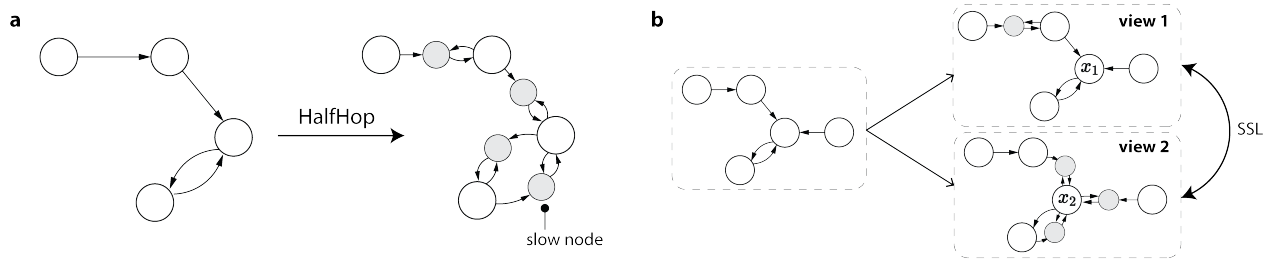

Figure 1. Overview of the Half-Hop augmentation. (a) On the left, we show an original graph and on the right, the graph after applying Half-Hop to all edges. We introduce slow nodes (gray) along each directed edge. (b) Half-Hop is used to generated diverse views for self-supervised learning methods. In the illustrated example, only the incoming edges of randomly selected nodes are half-hopped.

datasets show that by adding Half-Hop to a simple GCN backbone, we can achieve over 10% boost in performance, and when combining Half-hop with GraphSAGE (Hamilton et al., 2017), we achieve results that are comparable with state-of-the-art methods that use more complex architectures.

- *Study of Half-Hop and generalization analysis:* In both a theoretical investigation and empirical studies, we unpack the different ways that Half-Hop impacts learning. In particular, we analyze the impact of Half-Hop on the dynamics of message passing and show how our approach can slow down the effects of oversmoothing.

2. Method

In computer vision, increasing the resolution of an image (zooming in) is one of the most widely used augmentations for both supervised and unsupervised learning (Chen et al., 2020; Grill et al., 2020). Our idea is to rescale graphs in a similar fashion. Just as increasing the resolution of an image requires the introduction of new pixels, we upsample graphs by introducing new nodes along edges, and interpolating their node features based on the corresponding source and target nodes (see Figure 1). The modified graph can then be passed into any message passing-based neural network without further modification.

2.1. Background and Notation

Let $G = (V, E)$ denote our graph with nodes V and directed edges E. Each node $v_i \in V$ is associated with a set of d-dimensional features $x_i \in \mathbb{R}^d$. We note directed edge e_{ij} the edge going from node v_i to node v_j. In the message passing scheme, nodes exchange information with their neighboring nodes, through multiple rounds of message passing. Each node v_i updates its embedding by combining their ego-embedding (the node's embedding at a previous step) and the aggregated embeddings received from their neighbors. There are multiple implementations of the message passing layer, we highlight the GCN (Kipf & Welling, 2016) model, for which the output at layer ℓ is expressed as:

$$h_i^{(\ell)} = \sigma\left(\sum_{j \in \mathcal{N}(i) \cup \{i\}} \frac{1}{\sqrt{\hat{d}_j \hat{d}_i}} \mathbf{W}_\ell h_j^{(\ell-1)}\right),$$

where $\mathcal{N}(i)$ denotes the neighbors of v_i, $\mathbf{W}_\ell$ is a set of learnable shared weights used to compute messages, σ an activation function, and $\hat{d}_i = 1 + |\mathcal{N}(i)|$. Other GNN models include GraphSAGE which uses a different learnable matrix for the ego-embedding, and GAT (Veličković et al., 2017) which leverages attention during the aggregation step.

2.2. Half-Hop: Graph upsampling by inserting slow nodes between adjacent nodes

Half-hopping an edge in the graph. We consider a directed edge e_{ij} that is not a self-loop (i.e. $i \neq j$). To "half-hop" e_{ij}, we introduce a new node ν_k that splits the edge into two. The path $v_i \to v_j$ previously of length 1, is expanded: $v_i \to \nu_k \rightleftharpoons v_j$. Due to the added step, we refer to this new node as a "slow node". This modification to the graph can be expressed as follows:

$$\begin{cases} V' = & V \cup \{\nu_k\} \\ E' = & (E \setminus \{e_{ij}\}) \cup \{e_{i\to k}, e_{j\to k}, e_{k\to j}\}, \end{cases} \tag{1}$$

Note that both the source and target nodes communicate their messages to the slow node, but the slow node only passes information in the original direction. We find that this configuration is indeed optimal compared to other connectivity motifs (see ablations in Appendix B).

Interpolating slow node features. When constructing a slow node, we need to decide what features to assign to it, since zero or random initialization is not sufficient (Appendix B). A simple yet effective approach is to use linear interpolation of the source and target features. For edge $e_{i,j}$, we initialize the features of their slow node as follows:

$$\widetilde{x}_k = (1 - \alpha) x_j + \alpha x_i,$$

where α is some fixed scalar between 0 to 1. This parameter can be fine-tuned using a validation set. By adjusting α between 0 and 1, we can adjust the proximity of the slow node's initial features to the source or target node.

Half-hopping the graph. Now that we have established how an edge is modified, we can apply this operation to multiple edges in the graph. We can do this for all edges or for a subset of them. In our work here, we consider a node-level sampling to apply Half-Hop randomly on a subset of edges. For each node $v_i \in V$, with probability p, we "half-hop" all of v_i's incoming edges. Let $\mathcal{S}$ be the subset of nodes selected for half-hopping, $E_{\mathcal{S}}$ be the set of all directed edges that have target nodes in $\mathcal{S}$, where $N_s = |E_{\mathcal{S}}|$. The set of nodes in the new locally half-hopped graph is: $V' = V \cup \{\nu_k\}_{k=1}^{N_s}$.

To streamline notation, we can write a sample from this graph generative process as $(V', E') \sim hh_\alpha(G; p)$. When *all* edges in the graph are Half-Hopped (i.e. p=1), we will use an uppercase $\mathrm{HH}_\alpha(G)$. Note that in contrast to the node sampling generator, the fully half-hopped graph is a deterministic transformation.

2.3. Combining Half-Hop with any MPNN

After applying Half-Hop, we obtain a modified graph that can be passed into a message passing neural network. We treat original and slow nodes the same when it comes to MP, both nodes receive and send updates according to the same rules. We treat the Half-Hop operation as internal to the GNN, and any downstream operations such as loss computation or inclusion in further neural network encoders work only over the embeddings of the original nodes.

After training our model with Half-Hop, at inference time, we have the choice of whether to use the original graph G or the Half-Hopped graph $\mathrm{HH}_\alpha(G)$ as input. If we choose to not augment the graph at inference time, we will simply benefit from an improved model generalization, enabled by the use of Half-Hop as an augmentation during training (as we show in Section 4.4). If we do augment the graph, we would also take advantage of the improved MP enabled by Half-Hop (as we demonstrate in Section 3.2).

2.4. Using Half-Hop to generate views for Self-supervised learning

We next consider the use of Half-Hop for self-supervised representation learning. Self-supervised learning methods use augmentations to generate different views of a graph and then learn a representation space in which these different views are close to each other. With Half-Hop, we can create views where node i might be directly connected to its one-hop neighbors in the first, but half-hopped in the second. This generates a heterogeneous zooming and cropping into the neighborhood of different nodes. We expand on this in Section 3.2, where we show how Half-Hop alters the RF of the GNN.

Thus, we can generate two views $(\widetilde{G}_1, \widetilde{G}_2)$ of the graph to use as inputs to contrastive learning:

$$\widetilde{G}_1 \sim hh_\alpha(G; p_1), \widetilde{G}_2 \sim hh_\alpha(G; p_2)$$

where we parameterize the sampling procedure for each view with node-sampling probabilities p_1 and p_2. As explained in the previous section, when evaluating the contrastive loss, we only use the original nodes as positive/negative examples.

3. Understanding Half-Hop

In this section, we investigate Half-Hop from two angles. In Section 3.1, we study how Half-Hop alters the receptive field of the GNN model. In Section 3.2.2, we study the effect of Half-Hop on message passing from a spectral perspective. In both theory and experiments, we show that our approach slows down the smoothing process.

3.1. Reshaping the node's receptive field with Half-Hop

The receptive field (RF) of a model represents the parts of the input graph that have the most significant impact on the final embedding of a particular node. In MPNNs, the representation of a node is typically influenced by its neighboring nodes. In Half-Hop, the receptive field is reduced because messages are routed through intermediate nodes and thus 1-hop neighbors are no longer within 1-hop of one another. Thus, we wanted to understand how the RF is being shaped over rounds of message passing. To do this, we consider a simple 2D planar graph (Figure 2a) over which we apply multiple rounds of mean aggregation.

In Figure 2b, we visualize, for a given central node in this simple 2D graph, the contribution of a k-hop neighbor to the central node's final embedding. In the original graph (Fig. 2b, left), the weights of messages from nearest neighbors are quickly attenuated. Even after one step, the uniform weights between 1-hop neighbors and self-loops flattens the RF immediately from the central node. This smoothing process can often take place extremely quickly, as demonstrated in this example. When we apply Half-Hop (Fig. 2b, right), we show that the receptive field is altered: a more graceful decrease in the weight of neighbors is observed, with a higher amount of weight placed on the node itself. In Figure 2c, we plot the self-weight (y-axis) as a function of the message passing step (x-axis), and show that Half-Hop allows us to preserve the ego-embedding for nodes for longer. Even as we converge to large values of α, Half-Hop preserves a stable value for the self-weight due to the symmetry in mixing across source and target nodes. At the same

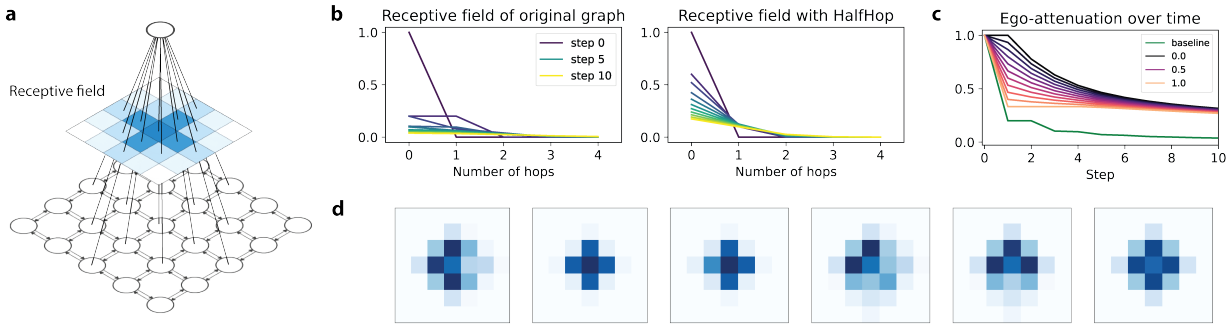

Figure 2. Analyzing how Half-Hop changes the receptive field (RF) of the GNN. Here we consider a 2D planar graph (a) and estimate the contribution of any node to the final embedding of the central node based on distance between the nodes (number of hops), without (b, left) and with Half-Hop (b, right). (c) The dynamics of attenuation of the ego-embedding for different values of α. (d) Example RFs obtained for probabilistic Half-Hop ($\alpha = 0.5$, $p = 0.75$) that highlight how the RF changes when different subsets of edges are half-hopped.

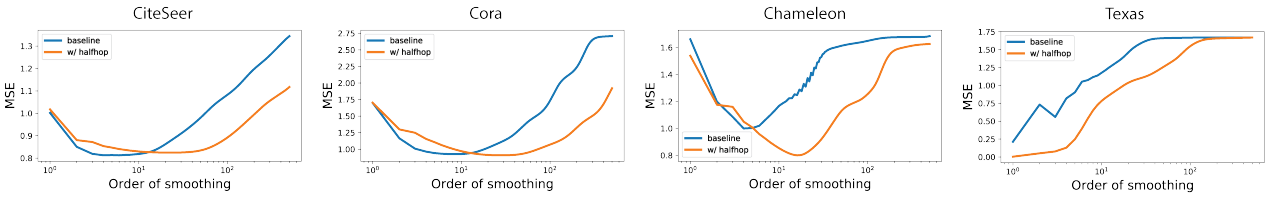

Figure 3. Analysis of isotropic diffusion with and without Half-Hop. Mean Squared Error (MSE) of linear ridge regression on the smoothed features after a number of rounds of message passing. From left to right, we illustarate it for CiteSeer, Cora, Chameleon and Texas.

time, most values of α have similar RFs when considering later stages of MP.

In self-supervised learning, we use Half-Hop to generate multiple views in which the same node can have different receptive fields. In Figure 2c, we highlight a few examples of receptive fields of the same central node when Half-Hop is applied with $\alpha = 0.5$ and $p = 0.75$ on the grid graph. Because we sample a subset of the nodes for half-hopping, we can generate diverse configurations. We find that the receptive field can be narrow, broad or a hybrid of both based on which nodes were half-hopped. This augmentation is reminiscent to zooming and cropping in the vision domain, in that we aggregate information at different scales.

3.2. Half-Hop shapes the dynamics of message passing

The dynamics of message passing depend on the original spectrum of the node features. To provide a rigorous characterization of the dynamics of smoothing, recent work (Keriven, 2022) provides new connections between the performance of GNNs and the covariance of the data. They show that, the covariance play a critical role in determining when over-smoothing starts to occur.

Thus, we hypothesized that by delaying message passing, Half-Hop could both slow down the point where over-smoothing occurs and also alter some properties of the message passing dynamics.

3.2.1. Diffusion on Real-World Datasets

To do this, we first use the evaluation framework proposed in (Keriven, 2022) to simulate the effect of diffusion on the Half-Hop graph vs. the original graph for four different real-world datasets (CiteSeer, Cora, Chameleon, Texas). In this case, messages are passed along the graphs using average pooling and no learned weights, and a final output layer is trained from a subset of nodes and tested on those held out unlabeled nodes. On the x-axis, we display the number of rounds of message passing and on the y-axis, the mean squared error (MSE) for the node regression task for all datasets. We note that CiteSeer/Cora are considered homophilic, while Chameleon/Texas are heterophilic.

When we compare the generalization dynamics for Half-Hop vs. the baseline, we find very different behavior for homophilic (left) vs. heterophilic graphs (right). In the homophilic graphs, we achieve overall similar rates of test error with the baseline and Half-Hop but we see the basin of low error solutions is widened (suggesting more stability) and the point where oversmoothing kicks in (MSE starts to go up again) is increased. Naturally, we would expect the factor to be at least twice as large, since Half-Hop divides the receptive field of the graph network by two, but we find that the transition point is even greater than what would be predicted by this factor.

When we examine heterophilic graphs (Chameleon, Texas),

we observe that Half-Hop achieves a significantly lower risk than the baseline. In the case of Chameleon, Half-Hop improves the descent and overall test risk significantly, this can be explained by the fact that heterophilic graphs suffer more from the oversmoothing, since we would be aggregating features of nodes that do not belong to the same class. This demonstrates the useful inductive bias in the Half-Hop model even without learning weights for message passing. In Texas, any rounds of mean aggregation (with learning of weights) hurt the test risk; However, Half-Hop achieves a much lower risk at the first few steps of message passing and also stabilizes the risk far longer. In Appendix F.1, we visualize how the embeddings changes over multiple rounds of MP, and show how Half-Hop also slows down the collapse of the latent space.

3.2.2. Analyzing the effect of Half-Hop on generalization

Our simulations suggest that Half-Hop changes the dynamics of message passing. Thus, we wanted to dig deeper and develop a result that allows us to compare the dynamics of message passing with and without Half-Hop. Throughout, we follow the assumptions and model described in (Keriven, 2022) (see Appendix A for more details).

A) Preliminaries and assumptions:

Graph Model. We adopt the latent space random graph model, where we assume that the observed node features $x_i = M^\top z_i$ are projections of some underlying latent features z_i, where M denotes an unknown projection matrix; We assume the latent features $z_i \sim \mathcal{N}(0, \Sigma)$, with covariance Σ. The edge weights are determined as a function of the latent variables by $W_{i,j} = \varepsilon + \exp(-\frac{1}{2}\|z_i - z_j\|_2^2)$, where ε is an unknown offset. The node labels, $y_i = z_i \top \beta^*$, are linear functions of the latent variable z_i with unknown coefficients β^*.

Objective. We consider a semi-supervised linear regression task where the goal is to estimate β^* and use it to predict the labels for nodes in the test set. Because only the projected node features x_i are observed, we can hope to recover the information by leveraging the connectivity of the graph. We use MSE to write the risk as $\mathcal{R}^{(k)} = \|Y_{te} - \widehat{Y}_{te}\|^2$, where Y_{te} is stacked labels for n_{te} unlabeled features and $\widehat{Y}_{te}$ are the estimated labels after k rounds of message passing.

B) How the spectrum impacts the dynamics of message passing: To present our main result, we first need to define the following function which we will use to approximate our test risk.

Definition 1. **(Keriven, 2022)** For a symmetric positive semi-definite matrix $S \in R^{d\times d}$, we define the function,

$$R_{\text{reg.}}(S) \overset{\text{def.}}{=} \left(\Sigma^{\frac{1}{2}}\beta^\star\right)^\top K \left(\Sigma^{\frac{1}{2}}\beta^*\right) \in \mathbb{R}_+$$

where $K = \left(\text{Id} - S^{\frac{1}{2}} M \left(\gamma \text{Id} + M^\top S M\right)^{-1} M^\top S^{\frac{1}{2}}\right)^2$, M is the projection matrix, β^* are the true model parameters, and γ is the ridge penalty in our least-squares estimator.

They show that the risk without message passing (MP) can be approximated by $\mathcal{R}^{(0)} \simeq R_{\text{reg.}}(\Sigma)$, and the risk after k rounds of MP can similarly be approximated as $\mathcal{R}^{(k)} \simeq R_{\text{reg.}}\left(A^{2k}\Sigma\right)$, where $A = (\text{Id} + \Sigma^{-1})^{-1}$. In this case, we can interpret A as a smoothing operator that is applied to the original spectrum, and interpret $\Sigma^{(k)} = A^{2k}\Sigma$ as the modified covariance after k rounds of MP.

C) Main Result: Our goal is to derive a similar style of result to understand how Half-Hop: (i) impacts the feature covariance of the embeddings, (ii) changes the rate of smoothing as we go deeper and execute more rounds of message passing. To do this, we want to derive an approximation of the risk in the form $R_{\text{reg.}}(\Sigma^{(\text{HH},k)})$ where $\Sigma^{(\text{HH},k)}$ is the approximated covariance of the node features after message passing with Half-Hop. We state our main result below and defer the proof to Appendix A.

Theorem 1. Message Passing Dynamics of Half-Hop. After $k \in \{1, 3, 5, \dots\}$ rounds of message passing, the risk obtained with *Half-Hop* can be approximated as:

$$\mathcal{R}^{(\text{HH},k)} \simeq R_{\text{reg.}}\left(\frac{1}{2}A^{k-1}\left(\text{Id} + ((1-\alpha)\text{Id} + \alpha A)^2\right)\Sigma\right).$$

The first term $A^{k-1}\Sigma$ smooths the original covariance a rate of $k-1$, which is roughly half the rate of smoothing without Half-Hop (2k). This in turn ensures that small eigenvalues decay at a slower rate. The rapid decay of small eigenvalues has been linked to over-smoothing. Half-Hop delays the point at which the smoothing stops being beneficial.

The second term in our augmented covariance, $((1-\alpha)\text{Id} + \alpha A)^2$, reveals the dependence on our mixing parameter α. In particular, we observe a uniform boost of the covariance spectrum coming from the $(1-\alpha)\text{Id}$; this implies that for small values of α we can interpret this as boosting of the self-loops and the minimum eigenvalues in the graph.

4. Experimental Results

In this section, we conduct a comprehensive empirical study of the effectiveness of Half-Hop on a wide range of datasets, models and learning paradigms.

4.1. Experimental Setup

Throughout, we test our approach using three of the most widely used graph models: the Graph Convolutional Network (GCN), GraphSAGE, and Graph Attention Network (GAT). In all of our experiments, we follow the same experimental setup as previous work. In the supervised experiments, we follow (Luo et al., 2022) in splitting the data into a development set and a test set. The hyperparameter

Table 1. Increase in supervised performance when using Half-Hop. The average and standard deviation of accuracy is computed over 20 random splits and model initializations. The absolute improvement (Δ) is also reported.

	Am. Comp.	Am. Photos	Co.CS	WikiCS
GCN	90.22 ± 0.60	93.59 ± 0.42	94.06 ± 0.16	81.93 ± 0.42
HH-GCN	90.92 ± 0.35	94.52 ± 0.22	94.71 ± 0.16	82.57 ± 0.36
Δ	**+0.70** (↑)	**+0.93** (↑)	**+0.65** (↑)	**+0.64** (↑)
GraphSAGE	84.79 ± 1.08	95.03 ± 0.33	95.11 ± 0.10	83.67 ± 0.45
HH-GraphSAGE	86.60 ± 0.49	94.55 ± 0.41	95.13 ± 0.21	82.81 ± 0.32
Δ	**+1.81** (↑)	**-0.48** (↓)	**+0.02** (↑)	**-0.86** (↓)

Table 2. Results on heterophilic graphs. We report the test accuracy across many datasets, and highlight the absolute improvement (Δ) in classification accuracy when the model is extended with Half-Hop. The "†" results are obtained from (Yan et al., 2021).

	Texas	Wisconsin	Actor	Squirrel	Chameleon	Cornell
Hom level	0.11	0.21	0.22	0.22	0.23	0.30
#Nodes	183	251	7,600	5,201	2,277	183
#Edges	295	466	26,752	198,493	31,421	280
#Classes	5	5	5	5	5	5
GCN†	55.14 ± 5.16	51.76 ± 3.06	27.32 ± 1.10	31.52 ± 0.71	38.44 ± 1.92	60.54 ± 5.30
HH-GCN	71.89 ± 3.46	79.80 ± 4.30	35.12 ± 1.06	47.19 ± 1.21	60.24 ± 1.93	63.24 ± 5.43
Δ	**+16.75** (↑)	**+19.04** (↑)	**+7.80** (↑)	**+15.67** (↑)	**+21.80** (↑)	**+2.70** (↑)
GAT†	52.16 ± 6.63	49.41 ± 4.09	27.44 ± 0.89	36.77 ± 1.68	48.36 ± 1.58	61.89 ± 5.05
HH-GAT	80.54 ± 4.80	83.53 ± 3.84	36.70 ± 0.92	46.35 ± 1.86	61.12 ± 1.83	72.70 ± 4.26
Δ	**+28.38** (↑)	**+34.12** (↑)	**+9.26** (↑)	**+9.58** (↑)	**+12.75** (↑)	**+10.81** (↑)
GraphSAGE†	82.43 ± 6.14	81.18 ± 5.56	34.23 ± 0.99	41.61 ± 0.74	58.73 ± 1.68	75.95 ± 5.01
HH-GraphSAGE	85.95 ± 6.42	85.88 ± 3.99	36.82 ± 0.77	45.25 ± 1.52	62.98 ± 3.35	74.60 ± 6.06
Δ	**+3.51** (↑)	**+4.70** (↑)	**+2.59** (↑)	**+3.64** (↑)	**+4.25** (↑)	**-1.35** (↓)
MixHop†	77.84 ± 7.73	75.88 ± 4.90	32.22 ± 2.34	43.80 ± 1.48	60.50 ± 2.53	73.51 ± 6.34
GGCN†	84.86 ± 4.55	86.86 ± 3.29	37.54 ± 1.56	55.17 ± 1.58	71.14 ± 1.84	85.68 ± 6.63
H_2GCN†	84.86 ± 7.23	87.65 ± 4.98	35.70 ± 1.00	36.48 ± 1.86	60.11 ± 2.15	82.70 ± 5.28
MLP†	80.81 ± 4.75	85.29 ± 3.31	36.63 ± 0.70	28.77 ± 1.56	46.21 ± 2.99	81.89 ± 6.40

tuning is performed using the development set only, and the accuracy of the best model on the development set is reported on the test set. For heterophilic datasets, we use the splits provided by (Pei et al., 2020), and also follow the same hyperparameter search protocol. For self-supervised benchmarks, we use the standard hyperparameters provided for each model and dataset (Zhu et al., 2020b; Thakoor et al., 2021). We provide more details in Appendix C.

4.2. Supervised node classification benchmarks

In our first set of experiments, we use a set of real-world benchmark datasets – Wiki-CS (Mernyei & Cangea, 2020), Amazon-Computers, Amazon-Photo (McAuley et al., 2015), and Coauthor datasets (Sinha et al., 2015). We train both GCN and GraphSAGE models with and without Half-Hop, and report the results in Table 4. Our results show that Half-Hop provides a good boost in performance across the datasets for the GCN, and GraphSAGE has more variability. HH-GraphSAGE on Amazon Computers is the most significant, where we observe a 1.81% improvement with Half-Hop over the baseline. These results provide evidence that Half-Hop can improve learning using only a simple augmentation of the graph.

4.3. Heterophilic benchmarks

We next applied Half-Hop to common real-world heterophilic benchmarks, including: Texas, Wisconsin, Actor, Squirrel, Chameleon and Cornell (Luan et al., 2022). On these datasets, we show improvements when we add Half-Hop to GCN, GraphSAGE and GAT. With both HH-GCN and HH-GAT, we see improvement of more than 10-20% on many of the datasets. To place these improvements in the context of more sophisticated methods for heterophilic graphs, we compare with: (i) MixHop (Abu-El-Haija et al., 2019), (ii) GGCN (Yan et al., 2021), and (iii) H_2GCN (Zhu et al., 2020a). See Figure 7 in the Appendix for a comparison of these different approaches and the techniques used to improve performance on heterophilic graphs.

When applied to standard GNNs, we find that Half-Hop boosts performance most significantly for the GCN and GAT, with modest improvements on the GCN of 2.7% on Cornell and 21.8% improvement on Chameleon. For the

Table 3. BGRL with different augmentation. Performance reported in terms of classification accuracy along with standard deviations. All experiments are performed over 20 random dataset splits and model initializations. At test time, the original graph is used.

	Augmentation	Am. Comp.	Am. Photos	Co.CS	Co.Phy	Wiki-CS
BGRL	None	87.12 ± 0.30	91.18 ± 0.38	91.85 ± 0.25	94.65 ± 0.11	78.69 ± 0.18
	FeatDrop + EdgeDrop	90.34 ± 0.19	93.17 ± 0.30	93.31 ± 0.13	95.73 ± 0.05	79.98 ± 0.10
	GCA	90.39 ± 0.22	93.15 ± 0.37	93.34 ± 0.13	95.62 ± 0.09	–
	Half-Hop	90.47 ± 0.25	93.18 ± 0.26	92.92 ± 0.11	95.69 ± 0.21	79.83 ± 0.53
	FeatDrop + EdgeDrop + Half-Hop	91.02 ± 0.27	93.88 ± 0.19	93.61 ± 0.13	95.75 ± 0.13	80.76 ± 0.71
GRACE	None	77.85 ± 0.96	88.47 ± 0.67	90.04 ± 0.36	OOM	70.61 ± 0.95
	FeatDrop + EdgeDrop	89.53 ± 0.35	92.78 ± 0.45	91.12 ± 0.20	OOM	80.14 ± 0.48
	GCA	87.85 ± 0.31	92.49 ± 0.09	93.10 ± 0.01	OOM	–
	Half-Hop	90.43 ± 0.28	93.58 ± 0.18	92.29 ± 0.12	OOM	79.86 ± 0.41
	FeatDrop + EdgeDrop + Half-Hop	91.11 ± 0.18	94.21 ± 0.26	93.59 ± 0.16	OOM	80.77 ± 0.40

Table 4. Increase in performance when using Half-Hop with different SSL frameworks. Performance reported in terms of classification accuracy along with standard deviations. All experiments are performed over 20 random dataset splits and model initializations.

	input (test)	encoder	Am. Comp.	Am. Photos	Co.CS	Co.Phy	WikiCS
GRACE	G	2-GCN	89.53 ± 0.35	92.78 ± 0.45	91.12 ± 0.20	OOM	80.14 ± 0.48
HH-GRACE	G	2-GCN	$\mathbf{91.11 \pm 0.18}$	94.21 ± 0.26	93.59 ± 0.16	OOM	79.77 ± 0.40
HH-GRACE	HH(G)	2-GCN	90.65 ± 0.19	$\mathbf{94.89 \pm 0.23}$	$\mathbf{94.76 \pm 0.14}$	OOM	80.15 ± 0.16
BGRL	G	2-GCN	90.34 ± 0.19	93.17 ± 0.30	93.31 ± 0.13	95.73 ± 0.05	79.98 ± 0.10
HH-BGRL	G	2-GCN	91.02 ± 0.27	93.88 ± 0.19	93.61 ± 0.13	95.75 ± 0.13	80.76 ± 0.71
HH-BGRL	HH(G)	2-GCN	90.94 ± 0.19	94.50 ± 0.35	$\mathbf{94.74 \pm 0.15}$	$\mathbf{96.13 \pm 0.10}$	80.37 ± 0.62
BGRL	G	3-GCN	90.04 ± 0.23	92.59 ± 0.34	92.42 ± 0.17	95.32 ± 0.51	78.22 ± 0.77
HH-BGRL	G	3-GCN	90.53 ± 0.27	93.09 ± 0.16	92.58 ± 0.20	95.45 ± 0.09	79.76 ± 0.61
HH-BGRL	HH(G)	3-GCN	$\mathbf{91.10 \pm 0.21}$	94.34 ± 0.25	$\mathbf{94.76 \pm 0.12}$	$\mathbf{96.10 \pm 0.09}$	$\mathbf{81.11 \pm 0.48}$

GAT, we see even larger improvements on Texas where we get a 28.38% boost with Half-Hop. The GraphSAGE encoder achieves the best performance out of the three models, and HH-GraphSAGE model reaches a performance that is comparable to the other competitor approaches that use more complex model components. We find that our approach provides an impressive gain for simple architectures that rivals with these other models without the need to define complex heuristics or very specialized architectures.

4.4. Self-supervised learning benchmarks

Graph representation learning methods rely on augmentations that are based on random transformations of the input. Thus, we can test the utility of Half-Hop for creating views for self-supervised learning and also as an add-on with existing augmentations. To do this, we combined Half-Hop with two state-of-the-art methods for self-supervised learning, BGRL (Thakoor et al., 2021) and GRACE (Zhu et al., 2020b). In this case, we report two sets of results: (i) Using Half-Hop as an augmentation during training and then applying the trained model to the original graph G and (ii) Applying Half-Hop to the full graph at inference time.

Using Half-Hop to generate views. First, we examined how Half-Hop could be combined with the existing augmentations used in BGRL and GRACE, and how well it performs as an augmentation on its own. We report the results in Table 3, where we show that by combining standard augmentations used in BGRL and GRACE with Half-Hop, we get a nice boost over the BGRL baseline and even more impressive improvement for the GRACE model (2% on Am. Photos and on Co. CS). Interestingly, when we use Half-Hop as an augmentation (without any Feature or EdgeDrop), we perform comparably to BGRL and adaptive edge-based augmentations in GCA.

Using Half-Hop at test time. As we observed in the supervised case, using Half-Hop at test time leads to improved message passing. We test HH-GRACE and HH-BGRL with both the original graph G and the half-hopped graph $\mathrm{HH(G)}$. Our results in Table 4 provide compelling evidence that Half-Hop improves self-supervised learning, in some cases by a significant margin. With HH-GRACE, we see an improvement of more than 2% on Amazon-Photos and Coauthor-CS. We also see similar improvements in HH-BGRL over the baseline which uses a 2-layer GCN.

When we make the GCN in BGRL deeper (3 layers), we find that the performance degrades by an average of 0.8%. When we do the same with HH-BGRL, we find that increase the depth leads to even higher performance. In particular,

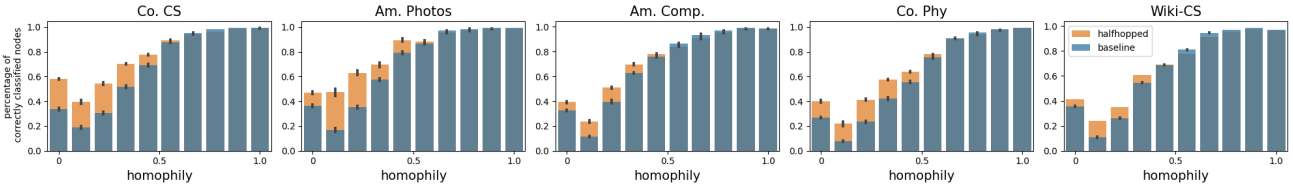

Figure 4. Percentage of correctly classified nodes for a given range of homophily, when using Half-Hop vs. the BGRL baseline.

we find a significant enhancement for HH-BGRL on the dataset with the lowest homophily (0.66), WikiCS, with added depth. To better understand the sources of these improvements, we breakdown the node-level accuracies by homophily and show that more heterophilic nodes are classified correctly (Figure 4).

5. Related Work

5.1. Graph data augmentations for regularization

Data augmentation is widely used in graph learning to improve the robustness and generalization capabilities of models. Thus, this has spurred a lot of interest in designing augmentations for graph-structured data (Zhu et al., 2021).

Feature perturbations. The most common feature-based augmentation is feature masking or feature dropout (You et al., 2020; Veličković et al., 2019), which involves randomly setting a subset of a node's features to zero. Other approaches like FLAG (Kong et al., 2022) and LA (Liu et al., 2022) use generative modeling to introduce gradient-based adversarial perturbations to the node's features. Mixup for graphs has also been proposed but usually requires a particular architecture (Verma et al., 2019), or a sub-graph sampling strategy (Wang et al., 2021).

Edge Perturbation. Adding and removing edges can also be used to perturb the connectivity of a node. In DropEdge (Rong et al., 2020; Feng et al., 2020; Veličković et al., 2019), each edge has a given probability of being removed. In GCA (Zhu et al., 2021), an edge is more or less likely to be removed based on the connectivity of its target node. GCC (Qiu et al., 2020) uses random walks to sample ego-networks around a central node. Approaches that add edges, on the other hand, typically require more guidance, like GAug (Zhao et al., 2020) which uses neural edge predictors to infer the probability of an edge.

k-Hop augmentations. Adding edges can also be used to connect more distant neighbors (k-hops away). This is typically achieved using a diffusion process (Li et al., 2019; Hassani & Khasahmadi, 2020). With this type of augmentation, we are effectively expanding the receptive field of the GNN, and are able to replicate the zoom-out operation that we know in images. On the other hand, our proposed augmentation, reduces the receptive field and allows a zoom-in operation in that sense.

5.2. Graph manipulation at inference time

When graphs are sparsely connected, highly heterophilic or have bottlenecks, graph rewiring techniques are used to correct the connectivity of nodes. SDRF (Topping et al., 2021) adds edges based on the Ricci curvature, while DIGL (Gasteiger et al., 2019) uses a diffusion process to add edges. NeuralSparse (Zheng et al., 2020) and GGCN (Yan et al., 2021) are supervised methods that use labels to learn to remove task-irrelevant edges that typically cause oversmoothing in heterophilic neighborhoods. (Ding et al., 2018) uses a generative modeling framework to create a small number of nodes that connect different subgraphs.

5.3. Self-supervised and contrastive learning methods

Many state-of-the-art approaches for self-supervised learning (SSL) on graphs use augmentations to create different views (i.e., positive examples) and then encourage the representations of both views to be close in the latent space. For instance, GRACE (Zhu et al., 2020b) uses a contrastive loss to encourage positive examples (a new graph with dropped edges and node features) to become closer to one another, while considering all other nodes to be far away. BGRL instead uses a score-based approach that doesn't explicitly incorporate negative examples into the loss (Thakoor et al., 2021). Most of these approaches use relatively simple augmentations, like node and edge dropout, to create views for learning. However, adaptive graph augmentations like those proposed in GCA (Zhu et al., 2021) that use topology-level and node-attribute-level augmentations like 'node centrality' measures can also be used to determine edges to mask.

6. Discussion

In this work, we introduced Half-Hop, a novel network-level augmentation for regularizing learning in graphs. We show that Half-Hop offers significant enhancements in MPNNs, especially in heterophilic conditions and in self-supervised learning. Our results on heterophilic datasets show that by adding Half-Hop to a simple GCN backbone, we can achieve over 10% boost in performance. When we couple Half-Hop with GraphSAGE, we obtain comparable results with state-of-the-art methods that use more complex losses or architectural modifications. In the future, it would be interesting to combine Half-Hop with approaches that use more complex architectures/losses to improve performance further.

References

Abu-El-Haija, S., Perozzi, B., Kapoor, A., Alipourfard, N., Lerman, K., Harutyunyan, H., Ver Steeg, G., and Galstyan, A. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *international conference on machine learning*, pp. 21–29. PMLR, 2019.

Alon, U. and Yahav, E. On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=i80OPhOCVH2.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. A simple framework for contrastive learning of visual representations. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 1597–1607. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/chen20j.html.

Cui, Z., Henrickson, K., Ke, R., and Wang, Y. Traffic graph convolutional recurrent neural network: A deep learning framework for network-scale traffic learning and forecasting. *IEEE Transactions on Intelligent Transportation Systems*, 21(11):4883–4894, 2019.

Ding, M., Tang, J., and Zhang, J. Semi-supervised learning on graphs with generative adversarial nets. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pp. 913–922, 2018.

Feng, W., Zhang, J., Dong, Y., Han, Y., Luan, H., Xu, Q., Yang, Q., Kharlamov, E., and Tang, J. Graph random neural networks for semi-supervised learning on graphs. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 22092–22103. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/fb4c835feb0a65cc39739320d7a51c02-Paper.pdf.

Gasteiger, J., Weißenberger, S., and Günnemann, S. Diffusion improves graph learning. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

Gaudelet, T., Day, B., Jamasb, A. R., Soman, J., Regep, C., Liu, G., Hayter, J. B., Vickers, R., Roberts, C., Tang, J., et al. Utilizing graph machine learning within drug discovery and development. *Briefings in bioinformatics*, 22(6):bbab159, 2021.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *International conference on machine learning*, pp. 1263–1272. PMLR, 2017.

Grill, J.-B., Strub, F., Altché, F., Tallec, C., Richemond, P., Buchatskaya, E., Doersch, C., Avila Pires, B., Guo, Z., Gheshlaghi Azar, M., et al. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in neural information processing systems*, 33:21271–21284, 2020.

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Hassani, K. and Khasahmadi, A. H. Contrastive multi-view representation learning on graphs. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 4116–4126. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/hassani20a.html.

Keriven, N. Not too little, not too much: a theoretical analysis of graph (over) smoothing. *arXiv preprint arXiv:2205.12156*, 2022.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Kong, K., Li, G., Ding, M., Wu, Z., Zhu, C., Ghanem, B., Taylor, G., and Goldstein, T. Robust optimization as data augmentation for large-scale graphs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 60–69, 2022.

Ktena, S. I., Parisot, S., Ferrante, E., Rajchl, M., Lee, M., Glocker, B., and Rueckert, D. Distance metric learning using graph convolutional networks: Application to functional brain networks. In *Medical Image Computing and Computer Assisted Intervention- MICCAI 2017: 20th International Conference, Quebec City, QC, Canada, September 11-13, 2017, Proceedings, Part I 20*, pp. 469–477. Springer, 2017.

Li, Z., Liu, Z., Huang, J., Tang, G., Duan, Y., Zhang, Z., and Yang, Y. Mv-gcn: Multi-view graph convolutional networks for link prediction. *IEEE Access*, 7:176317–176328, 2019. doi: 10.1109/ACCESS.2019.2957306.

Liu, S., Ying, R., Dong, H., Li, L., Xu, T., Rong, Y., Zhao, P., Huang, J., and Wu, D. Local augmentation for graph neural networks. In *International Conference on Machine Learning*, 2022.

Luan, S., Hua, C., Lu, Q., Zhu, J., Zhao, M., Zhang, S., Chang, X.-W., and Precup, D. Revisiting heterophily for graph neural networks, 2022. URL https://arxiv.org/abs/2210.07606.

Luo, Y., Luo, G., Yan, K., and Chen, A. Inferring from references with differences for semi-supervised node classification on graphs. *Mathematics*, 10(8), 2022. ISSN 2227-7390. doi: 10.3390/math10081262. URL https://www.mdpi.com/2227-7390/10/8/1262.

McAuley, J., Targett, C., Shi, Q., and van den Hengel, A. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pp. 43–52, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450336215. doi: 10.1145/2766462.2767755. URL https://doi.org/10.1145/2766462.2767755.

Mernyei, P. and Cangea, C. Wiki-cs: A wikipedia-based benchmark for graph neural networks. In *ICML Workshop on Graph Representation Learning and Beyond*, 2020.

Monti, F., Boscaini, D., Masci, J., Rodola, E., Svoboda, J., and Bronstein, M. M. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5115–5124, 2017.

Oono, K. and Suzuki, T. Graph neural networks exponentially lose expressive power for node classification. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1ldO2EFPr.

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

Pei, H., Wei, B., Chang, K. C.-C., Lei, Y., and Yang, B. Geom-gcn: Geometric graph convolutional networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1e2agrFvS.

Qiu, J., Chen, Q., Dong, Y., Zhang, J., Yang, H., Ding, M., Wang, K., and Tang, J. Gcc: Graph contrastive coding for graph neural network pre-training. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, KDD '20, pp. 1150–1160, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379984. doi: 10.1145/3394486.3403168. URL https://doi.org/10.1145/3394486.3403168.

Rong, Y., Huang, W., Xu, T., and Huang, J. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=Hkx1qkrKPr.

Sinha, A., Shen, Z., Song, Y., Ma, H., Eide, D., Hsu, B.-J. P., and Wang, K. An overview of microsoft academic service (mas) and applications. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15 Companion, pp. 243–246, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450334730. doi: 10.1145/2740908.2742839. URL https://doi.org/10.1145/2740908.2742839.

Stokes, J. M., Yang, K., Swanson, K., Jin, W., Cubillos-Ruiz, A., Donghia, N. M., MacNair, C. R., French, S., Carfrae, L. A., Bloom-Ackermann, Z., et al. A deep learning approach to antibiotic discovery. *Cell*, 180(4): 688–702, 2020.

Thakoor, S., Tallec, C., Azar, M. G., Azabou, M., Dyer, E. L., Munos, R., Veličković, P., and Valko, M. Large-scale representation learning on graphs via bootstrapping. *arXiv preprint arXiv:2102.06514*, 2021.

Topping, J., Di Giovanni, F., Chamberlain, B. P., Dong, X., and Bronstein, M. M. Understanding over-squashing and bottlenecks on graphs via curvature, 2021. URL https://arxiv.org/abs/2111.14522.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Veličković, P., Fedus, W., Hamilton, W. L., Liò, P., Bengio, Y., and Hjelm, R. D. Deep graph infomax. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rklz9iAcKQ.

Verma, V., Qu, M., Lamb, A., Bengio, Y., Kannala, J., and Tang, J. Graphmix: Regularized training of graph neural networks for semi-supervised learning. *CoRR*, abs/1909.11715, 2019. URL http://arxiv.org/abs/1909.11715.

Wang, Y., Wang, W., Liang, Y., Cai, Y., and Hooi, B. Mixup for node and graph classification. In *Proceedings of the Web Conference 2021*, pp. 3663–3674, 2021.

Yan, Y., Hashemi, M., Swersky, K., Yang, Y., and Koutra, D. Two sides of the same coin: Heterophily and oversmoothing in graph convolutional neural networks. *arXiv preprint arXiv:2102.06462*, 2021.

Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., and Leskovec, J. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 974–983, 2018.

You, Y., Chen, T., Sui, Y., Chen, T., Wang, Z., and Shen, Y. Graph contrastive learning with augmentations. *Advances in Neural Information Processing Systems*, 33: 5812–5823, 2020.

Zhao, T., Liu, Y., Neves, L., Woodford, O. J., Jiang, M., and Shah, N. Data augmentation for graph neural networks. *CoRR*, abs/2006.06830, 2020. URL https://arxiv.org/abs/2006.06830.

Zheng, C., Zong, B., Cheng, W., Song, D., Ni, J., Yu, W., Chen, H., and Wang, W. Robust graph representation learning via neural sparsification. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 11458–11468. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/zheng20d.html.

Zhu, J., Yan, Y., Zhao, L., Heimann, M., Akoglu, L., and Koutra, D. Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in Neural Information Processing Systems*, 33:7793–7804, 2020a.

Zhu, Y., Xu, Y., Yu, F., Liu, Q., Wu, S., and Wang, L. Deep graph contrastive representation learning, 2020b. URL https://arxiv.org/abs/2006.04131.

Zhu, Y., Xu, Y., Yu, F., Liu, Q., Wu, S., and Wang, L. Graph contrastive learning with adaptive augmentation. In *Proceedings of the Web Conference 2021*, pp. 2069–2080, 2021.

Appendix

A. Generalization Analysis

A.1. Problem Setup

We adopt the model developed in (Keriven, 2022) for our analysis. We invite the reader to refer to this amazing paper for more details. In this section, we add the necessary context (assumptions and notations) to set up our own result presented in the section A.2.

Semi-supervised node regression setup. We consider semi-supervised learning on an undirected graph of n nodes. We observe the entire graph, encoded by the adjacency matrix W, and the node features $x_1, x_2, \ldots, x_n$. Among the nodes, n_{tr} nodes' labels are observed while $n_{\text{te}} = n - n_{\text{tr}}$ nodes are without labels and used for test. Compactly, we stack the labels and features as rows of the matrices Y and X, and denote the training data as Y_{tr} and X_{tr} and the testing data as Y_{te} and X_{te}.

Instead of predicting the labels using the node features alone, inference can be improved by leveraging the connectivity of the graph. We consider a linear message passing neural network (MPNN). k rounds of message passing are applied according to the adjacency matrix W, which results in the *smoothing* of the node embeddings. The node labels are then predicted from the updated node emebddings. We consider ridge regression problem with respect to the features learned by the MPNN, where the estimator is given by

$$\hat{\beta}^{(k)} = \text{argmin}_{\beta} \frac{1}{2n_{\text{tr}}} \left\| Y_{\text{tr}} - X_{\text{tr}}^{(k)} \beta \right\|^2 + \gamma \|\beta\|^2, \tag{2}$$

where $X_{\text{tr}}^{(k)}$ are the feature embeddings of the training nodes after k rounds of message passing, and $\gamma > 0$ is a ridge penalty.

The test risk for our semi-supervised setting can be written as

$$\mathcal{R}^{(k)} = n_{\text{te}}^{-1} \left\| Y_{\text{te}} - X_{\text{te}}^{(k)} \hat{\beta}^{(k)} \right\|^2, \tag{3}$$

where $X_{\text{te}}^{(k)}$ are the feature embeddings of the test nodes after k rounds of message passing, and $\hat{\beta}^{(k)}$ is the estimator in (2) learned from the training data.

Before we present our analysis, we introduce the following technical function to simplify expressions:

Definition 1. (Keriven, 2022) For any symmetric positive semi-definite input matrix $S \in R^{d \times d}$, we define the function

$$R_{\text{reg.}}(S) \overset{\text{def.}}{=} \left(\Sigma^{\frac{1}{2}} \beta^\star\right)^\top K \left(\Sigma^{\frac{1}{2}} \beta^*\right) \in \mathbb{R}_+$$

where $K = \left(\text{Id} - S^{\frac{1}{2}} M \left(\gamma \text{Id} + M^\top S M\right)^{-1} M^\top S^{\frac{1}{2}}\right)^2$, and β^*, Σ are introduced in the model assumptions.

Previous risk results on ordinary MPNN. In (Keriven, 2022), they show that the risk associated with the ridge regression operator without any message passing of raw features can be approximated by

$$\mathcal{R}^{(0)} \simeq R_{\text{reg.}}(\Sigma),$$

the risk with k rounds of message passing is approximately,

$$\mathcal{R}^{(k)} \simeq R_{\text{reg.}}\left(A^{2k}\Sigma\right),$$

where $A = (\text{Id} + \Sigma^{-1})^{-1}$.

One key implication in these equations is that multiple rounds message passing induces a form of spectrum smoothing. As evidenced in above equations, the MPNN, with k rounds of message passing, effectively modifies the covariance of raw data features from Σ to $A^{2k}\Sigma$.

(Keriven, 2022) shows that analyzing the eigenvalues provides further insight into the smoothing phenomena. We note λ_i an eigenvalue of the covariance matrix Σ, and $\lambda_i^{(k)}$ the eigenvalues after k round of smoothing. (Keriven, 2022) shows that while large eigenvalues mostly maintain their magnitudes $\lambda_i^{(k)} \sim \lambda_i$, small eigenvalues decay exponentially $\lambda_i^{(k)} \sim \lambda_i^{2k+1}$. In other words, small eigenvalues decay faster than large eigenvalues. In case where β^* is aligned with eigenvectors with

small eigenvalues, this can be harmful smoothing. This framework enables us to understand how and when smoothing becomes harmful (over-smoothing). The rapid decay of small eigenvalues can be attributed to this phenomena. In the following, we show that Half-Hop slows down this decay and effectively enables the more graceful smoothing that we observe empirically in Figure 3.

A.2. Main Result

To analyze the generalization of the Half-Hop within the framework above, we use the directed variant of Half-Hop: $\mathrm{HH}^{(1)}$ described in Appendix B.1. In this variant, there is no backward edge from the target node to the slow node.

Theorem 1. The regression risks of $k \in \{1, 3, 5, \dots\}$ rounds of message passing with *Half-Hop* are:

$$\mathcal{R}^{(k)}_{\mathrm{HH}_\alpha} \simeq R_{\text{reg.}} \left(\frac{1}{2} A^{k-1} \left(\mathrm{Id} + ((1-\alpha)\mathrm{Id} + \alpha A)^2 \right) \Sigma \right),$$

where $A = (\mathrm{Id} + \Sigma^{-1})^{-1}$.

Through Theorem 1, we can inspect how Half-Hop manipulates the features. Large eigenvalues preserve most of their magnitude through message passing $\lambda^{(k)} \sim (1 + ((1-\alpha) + \alpha\lambda)^2)\lambda$ while small eigenvalues decay as $\lambda^{(k)} \sim (1 + (1-\alpha)^2)\lambda^{k+1}$. From these observations, we see that the decay rate of small eigenvalues is halved compared with ordinary MPNN without Half-Hop.

Proof. The basic idea behind our proof is similar to (Keriven, 2022), where we use matrix concentrations to approximate the node feature after multiple rounds of message passing. Following the proof of Theorem 4 in (Keriven, 2022), the regression risk of $\mathcal{R}^{(k)}_{\mathrm{HH}_\alpha}$ is approximated by $R_{\text{reg.}}(\Sigma')$ where Σ' approximates the covariance of the node features after k rounds of message passing. Hence, the proof boils down to calculating Σ'.

With Half-Hop, we introduce new nodes that we call slow nodes. In the augmented graph, there are two types of nodes: 1. the original nodes that we note x_{I} and 2. the slow nodes that we note x_{II}. We will use the superscript to denote the number of steps for message passing. In one step of message passing, the original nodes receive and aggregate messages from the slow nodes (we note this operation AGG). The slow nodes are updated based on the source node they are connected to. We can write this as:

$$x_{\mathrm{I}}^{(t+1)} = \mathrm{AGG}(x_{\mathrm{II}}^{(t)}), \quad x_{\mathrm{II}}^{(t+1)} = x_{\mathrm{I}}^{(t)}, \tag{4}$$

with initialization $x_{\mathrm{II}}^{(0)} = \alpha x_s^{(0)} + (1-\alpha) x_d^{(0)}$, where x_s and x_d represents the source and target nodes of the slow node. Now using Lemma 1 in (Keriven, 2022), we can say, with high probability, that the features after one round of message passing can be approximated as

$$x_{\mathrm{I}}^{(1)} \simeq \alpha A x + (1-\alpha)x, \quad x_{\mathrm{II}}^{(1)} = x. \tag{5}$$

The original nodes are updated based on a mixture of their self-embedding (x) and that of their neighbors (Ax). For example, let us look at two source nodes x_{s_1} and x_{s_2} connected to a target node x_d. Then the slow node features between the two source and target node are $\alpha x_{s_1} + (1 - \alpha x_d)$ and $\alpha x_{s_2} + (1 - \alpha x_d)$. So after AGG, the feature of the target node will be $\alpha(\frac{x_{s_1} + x_{s_2}}{2}) + (1-\alpha)x_d$. On the other hand, the slow nodes simply copy the features of their source nodes.

For a second round of message passing, we apply the same process again:

$$x_{\mathrm{I}}^{(2)} = AGG(x_{\mathrm{II}}^{(1)}) \simeq Ax, \quad x_{\mathrm{II}}^{(1)} = x_{\mathrm{I}}^{(1)} = \alpha A x + (1-\alpha)x, \tag{6}$$

where in the update of $x_{\mathrm{II}}^{(2)}$ we again apply the deterministic approximation to AGG by matrix multiplication.

We can now apply the recursive formulas (4) iteratively. This assumes that we can approximate the AGG operation with a multiplication of A for Half-Hop. This is due to the underlying model assumptions which imply that the node feature distribution remains the same after a linear combination of i.i.d. Gaussian variables and the node features are approximately independent when n is large. Hence, if we continue applying the recursive formula and replace each AGG operation over x by Ax, then for any $k \geq 0$, mathematical induction yield

$$x_{\mathrm{I}}^{(2k+1)} = \alpha A^{(k+1)} x + (1-\alpha) A^{(k)} x, \; x_{\mathrm{II}}^{(2k+1)} = A^k x.$$

Now we complete the proof by recalling that the covariance of the generic node feature x is Σ and hence

$$\Sigma' = \frac{1}{2} A^{k-1} \left(\mathrm{Id} + ((1-\alpha)\mathrm{Id} + \alpha A)^2 \right) \Sigma. \tag{7}$$

Remark. The modified covariance term in Equation 7 consists of two main terms, the first being a component that smooths the original covariance with A at a rate of $k-1$, which is roughly half the original rate of smoothing for the graph without Half-Hop. In addition to this first slower smoothing term, we also find a second contribution to the new covariance. The second modified smoothing term is $((1-\alpha)\mathrm{Id} + \alpha A)^2$, where in this case we see a uniform boosting of the covariance spectrum coming from the first term and weighted by (1-α), and a second term coming from a rescaling of A. Thus, for small values of α we can interpret this as having a strong boosting of the self-loops in the graph. We also confirm that this is indeed the case for our analysis of the receptive field for different values of α in Figure 3.

B. Ablations

B.1. Testing the directionality of edges added in Half-Hop

Recall that for an edge e_{ij}, Half-Hop introduces a new slow node ν along the edge from v_i to v_j as follows:

$$\mathrm{HH}: \quad v_i \rightarrow \nu \leftrightarrow v_j.$$

In this experiment, we try alternative connectivity schemes. In the first variant, $\mathrm{HH}^{(1)}$, we do not introduce a backward edge that goes from the destination to the slow node:

$$\mathrm{HH}^{(1)}: \quad v_i \rightarrow \nu \rightarrow v_j.$$

The second variant, $\mathrm{HH}^{(2)}$, we add an edge going from the slow node to the source node:

$$\mathrm{HH}^{(2)}: \quad v_i \leftrightarrow \nu \leftrightarrow v_j.$$

We assess the influence of the above connectivity schemes for the vanilla GCN model across three different datasets - Texas, Actor and Cornell, and we tabulate our results in Table 1. Significantly better performance can be noted for our proposed connectivity scheme HH, which also happens to be the more intiuitive solution since it respects the directionality of the original edge, and the slow node gets to communicate with both source and target.

The training scheme, hyperparameters, and the model used for this experiment are the same as that of the results reported in the paper for heterophilic datasets.

Dataset	HH $v_i \rightarrow \nu \leftrightarrow v_j$	$\mathrm{HH}^{(1)}$ $\mathrm{v}_i \rightarrow \nu \rightarrow v_j$	$\mathrm{HH}^{(2)}$ $\mathrm{v}_i \leftrightarrow \nu \leftrightarrow v_j$
Texas	71.71 ± 8.76	68.8 ± 6.50	58.47 ± 5.56
Actor	33.35 ± 1.00	32.17 ± 0.84	31.93 ± 1.26
Cornell	63.42 ± 5.62	57.66 ± 6.89	42.16 ± 6.57

Table 1. Ablations of different connectivity motifs for slow nodes on heterophilic datasets. The three connectivity motifs are displayed along the columns, from left to right, Half-Hop as proposed, $\mathrm{HH}^{(1)}$ which still introduces delay but doesn't add a backward edge from the target node to slow node, and $\mathrm{HH}^{(2)}$ which doesn't have two bi-directional edges and hence doesn't have the same delayed message passing effect. The performance values correspond to that of vanilla GCN model coupled with the respective HH variant.

B.2. Testing different slow node initializations

In this experiment, we ablate the linear interpolation initialization we use for the slow node (Section 2.2). We test two simpler alternatives: 1) 'Zero': Setting the initial features to zero, 2) 'Random': Initializing the with random values in the range of [0,1).

As tabulated in Table 2, from the drop in performance as we move from 'Weighted Sum' to 'Zero' to 'Random' we can clearly observe a significant gain upon using our proposed 'Weighted Sum' scheme. This would be as the alpha parameter

in the weighted sum scheme enables the new slow node to be initialized with a blend of information from the source and target node embeddings.

For this experiment too we use the same hyperparameters, training scheme, and model as used for the results reported in the paper for heterophilic datasets.

Dataset	Weighted Sum	Zeros	Random
Texas	72.88 ± 7.17	61.8 ± 5.91	53.33 ± 5.27
Actor	33.39 ± 1.29	28.93 ± 2.83	24.7 ± 1.16
Cornell	63.33 ± 5.70	49.55 ± 7.06	37.48 ± 6.93

Table 2. Ablation of weighted sum method of initializing slow node embeddings The performance values correspond to that of vanilla GCN model coupled with HH utilizing respective node embedding initialization

C. Details for Supervised Experiments

For comprehensive analysis on the full-spectrum of real-world graphs, we study the effects of our approach on both homophilic and heterophilic datasets.

Homophily. We follow the definition of *node homophily ratio* as used in (Pei et al., 2020) given by the formula:

$$\frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} \frac{|\{(v, w) : w \in \mathcal{N}(v) \wedge y_v = y_w\}|}{|\mathcal{N}(v)|}$$

In the above formula - $\mathcal{V}$ denotes the set of all nodes in the graph, $\mathcal{N}(v)$ denotes all the neighbors of an arbitrary node v, and y_v denotes the class membership of the node $v \in \mathcal{V}$.

In our experiments, we treat datasets with Homophily Level ≥ 0.5 as *Homophilic datasets* and rest as *Heterophilic datasets*.

C.1. Homophilic Datasets

Amazon Computers and Amazon Photos (McAuley et al., 2015): Graphs pertaining to Amazon's co-purchasing data. Products (nodes) are linked with each other if they are frequently bought together. Each node's features are the product's word embeddings (Bag of words). The task is to classify the nodes into product categories.

Coauthor CS and Coauthor Physics (Sinha et al., 2015): Nodes represent word embeddings (Bag of words) of particular keywords used by authors in their articles, and edges indicate collaboration between authors. The task is to categorize the authors into particular research fields based on their keywords and collaborations.

WikiCS (Mernyei & Cangea, 2020): The network graph of Computer Science Wikipedia articles. Each node is an article with features as word embeddings (GLoVe) of the article content and the edge represents the link. The task is to classify the node to one of 10 article categories.

Basic statistics of the homophilic datasets used are listed in Table 3. The experimental setup follows that of (Luo et al., 2022), where we split the dataset into development and test sets. All the hyperparameter tuning is done on the development set and the best models are evaluated on the test set. The runs are averaged over 20 random splits to minimize noise. We follow a 60:20:20% train/val/test split for the Amazon and Coauthor datasets, and 20 pre-split masks provided in the WikiCS dataset.

C.2. Heterophilic Datasets

Texas, Wisconsin, Actor, Chameleon & Cornell (Luan et al., 2022): five real-world datasets with graphs that have a homophily level ≤ 0.30. Basic statistics of the datasets are listed in Table 4.

We full the experimental setup in (Pei et al., 2020), we use the same 10 train/val/test splits that are provided. We also perform similar hyperparameter tuning using randomized grid search using only the train and validation sets. Once we find the best model, we report the accuracy on the test set, which is only seen once. We include the final hyperparameters of the best models for each architecture and dataset in Table 5.

Table 3. Statistics of homophilic datasets used in our experiments

	Nodes	Edges	Features	Classes	Node Homophily Ratio
Amazon Photos	7,650	119,081	745	8	0.8365
Amazon Computers	13,752	245,861	767	10	0.7853
Coauthor CS	18,333	81,894	6,805	15	0.8320
Coauthor Physics	34,493	247,962	8,415	5	0.9153
Wiki CS	11,701	216,123	300	10	0.6588

Table 4. Statistics of heterophillic datasets used in our experiments

	Nodes	Edges	Classes	Node Homophily Ratio
Texas	183	295	5	0.11
Wisconsin	251	488	5	0.21
Film	7,600	26,752	5	0.22
Squirrel	5,201	198,493	5	0.22
Chameleon	2,277	31,421	5	0.23
Cornell	183	280	5	0.30

D. Details for Self-supervised Experiments

We use the same real-world datasets (Amazon-Photos, Amazon-Computers, Coauthor-CS and Coauthor-Physics) as the ones used in supervised setting. The whole dataset graph is used to train the self-supervised learning model as a transductive task, and for the linear evaluation phase, the dataset is split into train/val/test in the (10:10:80%) format respectively following the same setup as (Thakoor et al., 2021).

When using GRACE (Zhu et al., 2020b) and BGRL (Thakoor et al., 2021), we refer to the hyperparamters reported in their respective works. The edge-dropout and feature-dropout hyperparameters used are *edge-masking probability* for both the views ($p_{e,1}$, $p_{e,2}$), and *feature-masking probability* for both the views ($p_{f,1}$, $p_{f,2}$). The Half-Hop hyperparameters used for the training are *Half-Hop probability* for both the views ($p_{hh,1}$, $p_{hh,2}$) and the coefficient used for combining features for both the views (α_1, α_2). Details of their values are tabulated in Table 6.

The method of evaluation follows the linear evaluation protocol (Veličković et al., 2019), where the learned weights of the self-supervised learning model are frozen and the output representations are used to tune and train a Linear Classifier (without propagating gradients to the encoder). We use an l_2-regularized Logistic Regression Model with a *liblinear* solver from the Scikit-learn library (Pedregosa et al., 2011).

E. Comparisons with other GNNs

There have been multiple modifications on top of traditional GNN architectures to optimize for the task of heterophilic node classification. In Table 7 we detail the different architectural components, losses, and pieces of information that are used to improve performance for heterophilic datasets. In the table, we breakdown the different components of popular methods that we compare with in the main text, including: MixHop (Abu-El-Haija et al., 2019), (ii) GGCN (Yan et al., 2021), and (iii) H_2GCN (Zhu et al., 2020a).

F. Additional Experiments

F.1. Latent Visualization across GNN Layers

In this experiment, we visualize the latent space of node embeddings across various depths, with and without Half-Hop (Figures 1, 2 and 3) using a GCN and GraphSAGE encoder.

Observing the latent space visualizations, we can make a few interesting observations. Upon applying Half-Hop, the embeddings of the same class appear to aggregate together better while the embeddings of different classes seem to maximally distance themselves from each other and from the center of the latent space. Though this is slightly observed in

Table 5. Heterophilic Datasets.

Dataset	Model	lr	weight decay	depth	hidden	dropout	α	p
Texas	HH-GCN	0.0291	0.0096	2	64	0.8058	0.0043	0.9526
	HH-GraphSAGE	0.0170	0.0053	2	64	0.1967	0.9397	0.7140
	HH-GAT	0.0328	0.0066	2	32	0.1288	0.0902	0.9841
Wisconsin	HH-GCN	0.0105	0.0002	3	128	0.6612	0.9937	0.7140
	HH-GraphSAGE	0.0202	0.0042	3	64	0.3462	0.0100	0.6177
	HH-GAT	0.0539	0.0068	3	16	0.2141	0.0026	0.9797
Actor	HH-GCN	0.0313	0.0087	3	64	0.5511	0.0369	0.5466
	HH-GraphSAGE	0.0133	0.0090	3	32	0.3737	0.0116	0.8368
	HH-GAT	0.0009	0.0001	3	128	0.8708	0.0549	0.9594
Squirrel	HH-GCN	0.0053	0.0001	3	128	0.2455	0.0145	0.8257
	HH-GraphSAGE	0.0296	0.0001	2	128	0.8668	0.9474	0.5198
	HH-GAT	0.0027	0.0001	3	64	0.5131	0.9277	0.1549
Chameleon	HH-GCN	0.0318	0.0057	2	128	0.8040	0.0510	0.9986
	HH-GraphSAGE	0.0225	0.0001	2	32	0.7175	0.9834	0.6226
	HH-GAT	0.0012	0.0008	3	64	0.0439	0.9766	0.9386
Cornell	HH-GCN	0.0505	0.0055	2	32	0.4123	0.0145	0.9660
	HH-GraphSAGE	0.0697	0.0018	2	64	0.0697	0.8807	0.5660
	HH-GAT	0.0572	0.0070	2	64	0.0572	0.0710	0.9979

Table 6. Hyperparameter settings for unsupervised BGRL learning with Half-Hop

Aug. Hyperparameters	Am. Comp.	Am. Photos	Co. CS	Co. Phy	WikiCS
$p_{hh,1}$	0.75	0.75	0.75	0.75	0.75
$p_{hh,2}$	0.75	0.75	0.75	0.75	0.75
α_1	0.50	0.50	0.50	0.50	0.50
α_2	0.50	0.50	0.50	0.50	0.50
$p_{f,1}$	0.20	0.10	0.30	0.10	0.20
$p_{f,2}$	0.10	0.20	0.40	0.40	0.10
$p_{e,1}$	0.50	0.40	0.30	0.40	0.20
$p_{e,2}$	0.40	0.10	0.20	0.10	0.30

	Higher-order neighbors	Weights for self-loops	Concat across layers	Dynamic gating
GCN	✗	✗	✗	✗
SAGE	✗	✓	✗	✗
MixHop	✓	✗	✓	✗
GGCN	✗	✗	✗	✓
H_2GCN	✓	✓	✓	✗
HH-GCN	✗	✗	✗	✗
HH-SAGE	✗	✓	✗	✗

Table 7. Different components used in graph neural networks optimized for heterophilic node classification. From left to right, we show methods that incorporate additional information from higher-order neighbors, separate weights for self-loops, and other additional components. Here, we observe the fact that Half-Hop is lightweight and doesn't require extra components in the loss and also doesn't explicitly compute separate weights for self-loops.

all cases, this can be clearly noted in the case of GraphSAGE + Half-Hop for the Citeseer dataset in Figure 3. Visualizations of latents for vanilla GCN (without Half-Hop) indicate poor class separation (Figure 1 and Figure 2) and this can be seen reflected directly in vanilla GCN's performance as noted in Table 2 in the paper. On a more general note, class separation appears best after the second and third layers as observed across Figures 1, 2 and 3. This is similar to what is observed in

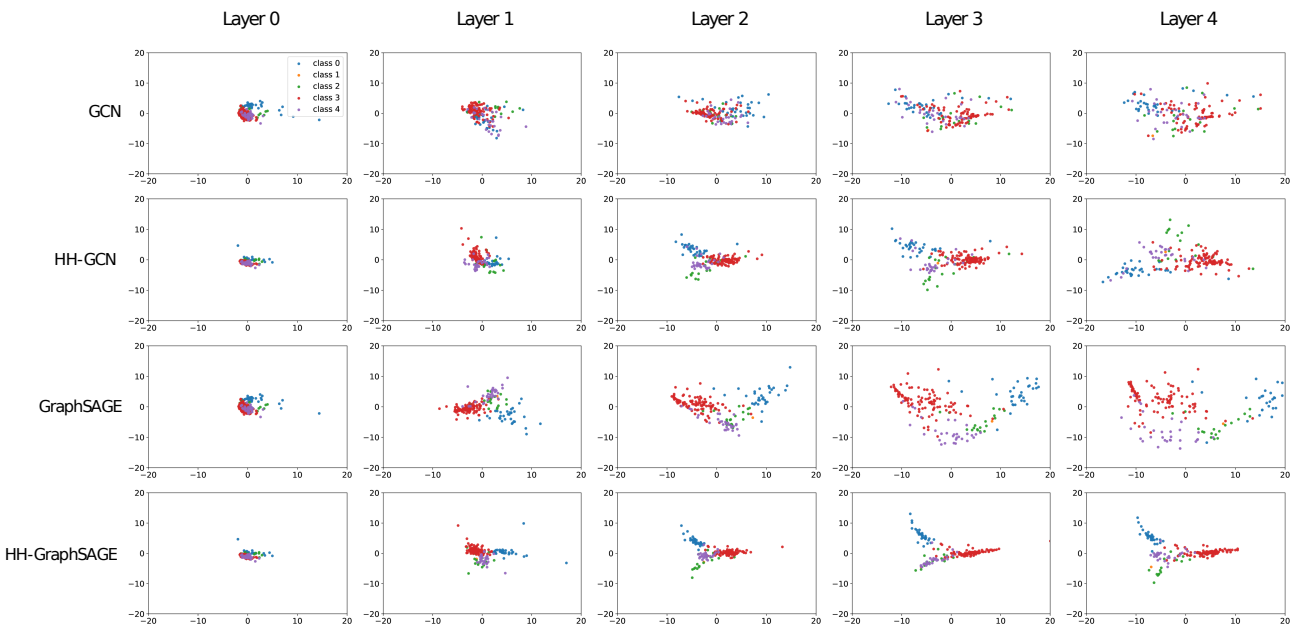

Figure 1. Latent space visualizations with and without Half-Hop for Texas dataset. We can note how similar data points cluster closer together in the cases where Half-Hop is applied. As class 1 is very rare, it is not clearly visible in the visualization. 'Layer 0' denotes the latent space before the GNN layers are applied.

terms of raw performance - the best-performing models are 2 or 3 layers deep.

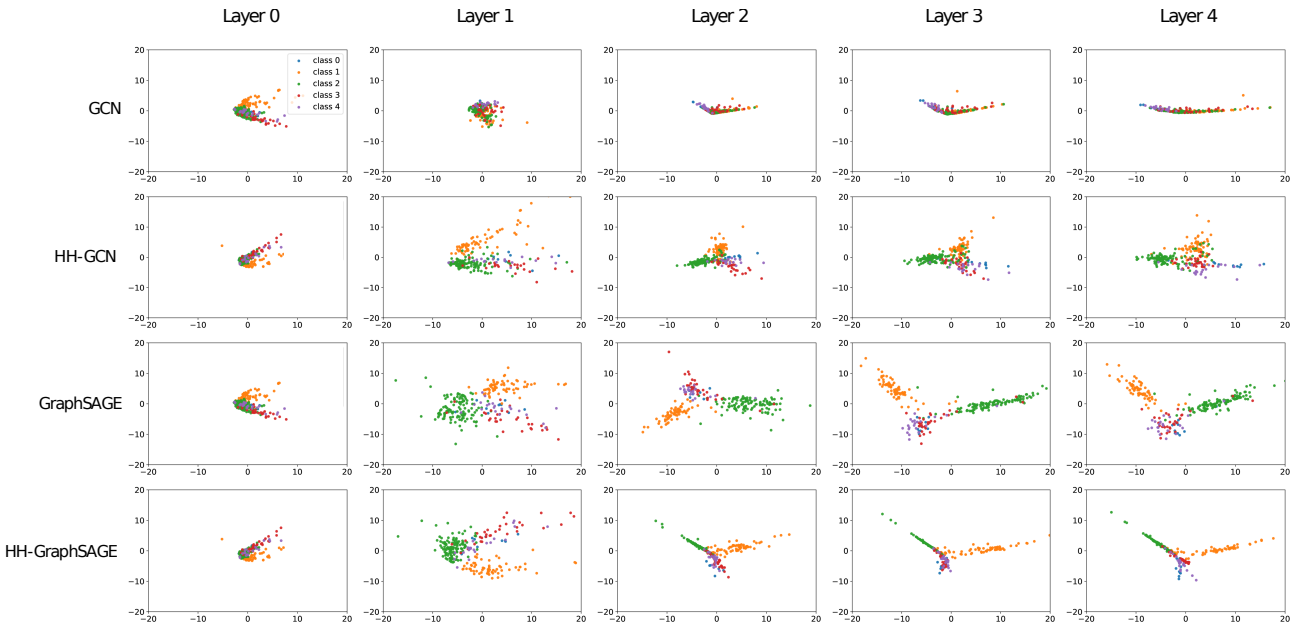

Figure 2. Latent space visualizations with and without Half-Hop for Wisconsin dataset. Though the color scheme is the same, it appears different from that of Figure 1 due to the difference in the distribution of nodes across classes in Texas and Wisconsin datasets.

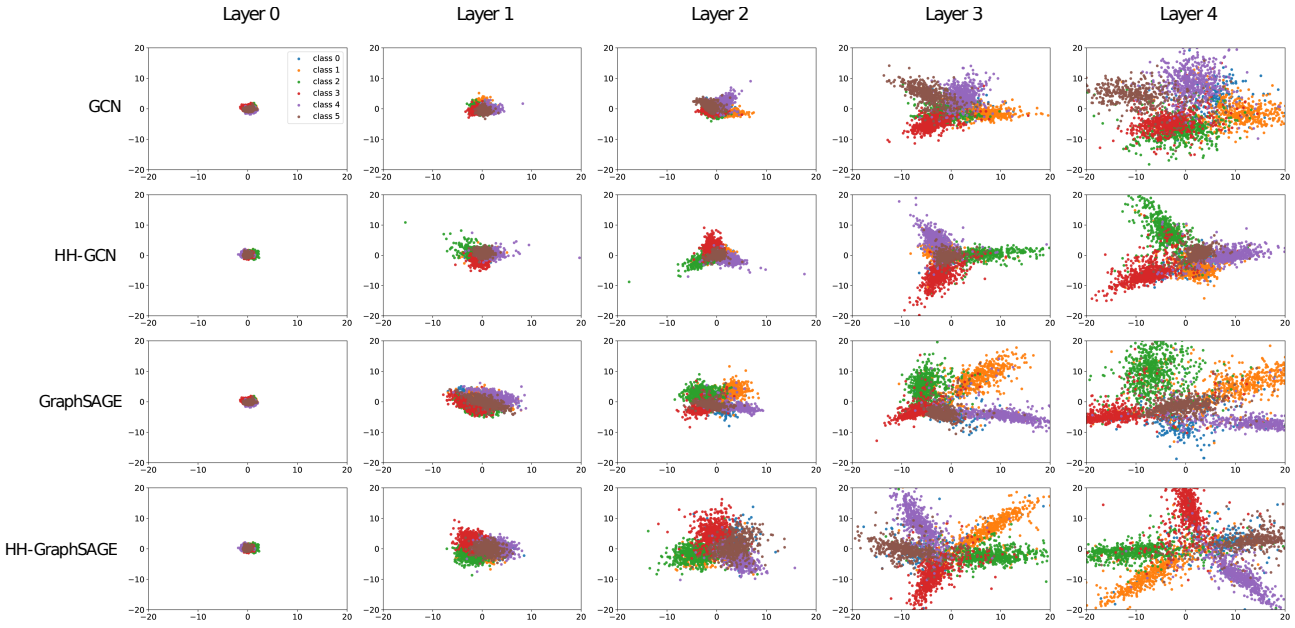

Figure 3. Latent space visualizations with and without Half-Hop for Citeseer dataset. We can again note how in the cases where Half-Hop is applied, similar latents seem to cluster closer together. Citeseer has a larger number of nodes compared to Texas and Wisconsin datasets and hence the latents in these visualizations appear to be packed more densely in comparison.