LINEAR GRAPH NEURAL NETWORKS FOR ALL

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph neural networks (GNN) have received a lot of attention in recent years due to their generalizability for graph-related tasks. Contrary to the general tendency to higher expressiveness and complexity, recent works show that linear GNNs can achieve comparable accuracy to nonlinear models in node-level tasks, where high complexity can easily result in overfitting. Such a success of linear GNNs implies that nonlinearity is not essential for GNNs, and we can understand existing models better if we remove their nonlinearity; we focus on how they propagate node-wise information to the entire graph. In this work, we formally define a linear GNN in relation to logistic regression and conduct systematic linearization on a variety of existing models. The observations derived from linearization guide us to propose SLIP, a new linear GNN that is more interpretable than existing models, successes in every unit test designed to cover all possibilities of graph datasets, and achieves comparable accuracy to nonlinear models in homophily and heterophily graphs.

1 INTRODUCTION

Graph neural networks (GNN) are popular approaches for various graph-related tasks such as node classification, link prediction, or graph generation to name a few. GNNs learn low-dimensional representations of nodes by combining the information of node features and a graph structure, providing rich semantic information to solve each task by end-to-end training. Each layer of a GNN performs feature transformation, which updates the current representations of nodes in a node-wise way, and propagation (or aggregation), which combines the representations of multiple nodes.

Recent works (Wang et al., 2021; Zhu & Koniusz, 2021; Li et al., 2022) show that GNNs consisting of only linear components perform incredibly well on real-world graphs. This implies that the nonlinearity of feature transformation is not essential for achieving high accuracy as long as node-level information is propagated effectively to distant neighbors in a graph. The strength of linear models is notable in semi-supervised learning, where the amount of training data is insufficient to learn the many parameters in complex GNN models. In such cases, the high complexity of models increases the risk of overfitting to any bias or noise existing only in training data, decreasing accuracy.

The success of linear models on graph-related tasks poses a new question to recent works on GNNs: *how are they different from each other philosophically if we remove all nonlinear components?* That is, if nonlinearity is optional (or even redundant in some cases), we can understand and compare the core mechanisms of GNNs by *linearizing* their propagator functions. The spectral analysis of graph convolutions (Zhu & Koniusz, 2021; Li et al., 2022) is related to this problem as it provides a way to compare different graph convolution operators in the spectral view, but it has a limitation that the result of analysis is not practical in terms of actual usage on real-world graphs.

In this work, we first define a *linear GNN* based on the relationship with logistic regression, because the goal of linearization is not just making a linear decision with respect to features, but providing global interpretability on a decision process; linearity with features is not a sufficient condition to get the global interpretability if the propagator function is too complex or contain learnable parameters. Then, we perform systematic linearization on different GNNs to understand and compare their core mechanisms in terms of feature propagation. The result not only provides a new way of comparing different GNNs in essence, but also gives us notable insights to design our own approach.

We then propose SLIP (Sparse Linear Propagation), our novel linear GNN for node classification on both homophily and heterophily graphs. SLIP has three main advantages. First, SLIP is accurate, as it combines the features propagated with different numbers of steps. Second, SLIP is interpretable,

as it makes linear decisions while limiting the complexity of feature transformation. Finally, SLIP is robust, since it removes unnecessary feature components by performing sparse feature selection.

The performance of SLIP is validated in two types of experiments. First, SLIP works well in every *unit test* we propose in this work. These unit tests are carefully designed to cover all possibilities of graph datasets based on the mutual information between features, labels, and a graph structure, and can be used as a controlled testbed to test any new graph algorithm. Second, SLIP shows comparable to or even higher accuracy than those of nonlinear models in real-world graphs of both homophily and heterophily. SLIP also provides global interpretability to measure feature importances.

Our contributions are summarized as follows:

- We formally define a *linear GNN* in the framework of logistic regression and compare the propagation functions of existing GNN models through *linearization* (in Section 3).
- We categorize the possible types of real-world graphs and provide comprehensive *unit tests* that any graph model can utilize for the robust and effective design (in Section 4).
- We propose SLIP, a linear GNN model that successes in all unit tests and real-world graphs even with a simple interpretable design of feature propagation (in Section 5).
- We run extensive experiments on homophily and heterophily graphs to highlight the failure of existing GNNs and the superiority of our proposed SLIP (in Section 6).

2 Problem Definition and Related Works

We introduce the problem definition of semi-supervised node classification, symbols frequently used in this paper, and related works on graph neural networks (GNN).

Problem definition We are given a graph $G = (\mathbf{A}, \mathbf{X})$, where $\mathbf{A} \in \mathbb{R}^{n \times n}$ is an adjacency matrix, $\mathbf{X} \in \mathbb{R}^{n \times d}$ is a matrix of node features, n is the number of nodes, and d is the number of features. Then, *semi-supervised node classification* is to train a classifier f from a vector $\mathbf{y} \in [1, c]^m$ of labels observed for m training nodes, where $m \ll n$, and c is the number of target classes.[1]

We use the following symbols to represent modified adjacency matrices. $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is the adjacency matrix with self-loops. $\tilde{\mathbf{D}} = \tilde{\mathbf{A}}\mathbf{1}_{n \times 1}$ is the diagonal degree matrix of $\tilde{\mathbf{A}}$, where $\mathbf{1}_{n \times 1}$ is the matrix of size $n \times 1$ filled with ones. $\tilde{\mathbf{A}}_{\text{sym}} = \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$ is the symmetrically normalized $\tilde{\mathbf{A}}$. Similarly, $\mathbf{A}_{\text{sym}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ is also a symmetrically normalized $\mathbf{A}$ without self-loops. There are other types of normalization $\mathbf{A}_{\text{row}} = \mathbf{D}^{-1}\mathbf{A}$ and $\mathbf{A}_{\text{col}} = \mathbf{A}\mathbf{D}^{-1}$ (and accordingly $\tilde{\mathbf{A}}_{\text{row}}$ and $\tilde{\mathbf{A}}_{\text{col}}$), which we call row and column normalization, respectively, based on the direction of normalization.

Graph neural networks There are a massive number of GNNs proposed with different motivations and insights. Recent surveys (Zhou et al., 2020; Wu et al., 2021) on GNNs introduce a way to categorize existing models into a few groups including spectral models (Defferrard et al., 2016; Kipf & Welling, 2017), sampling-based models (Hamilton et al., 2017; Ying et al., 2018), attention-based models (Velickovic et al., 2018; Kim & Oh, 2021; Brody et al., 2022), and deep models with residual connections (Li et al., 2019; Chen et al., 2020). GNNs are often fused with graphical inference for fully exploiting observed labels at test time (Yoo et al., 2019; Huang et al., 2021).

There are recent works (Xu et al., 2019; Garg et al., 2020; Papp et al., 2021) that aim to improve the expressiveness of GNNs in comparison to the Weisfeiler-Lehman (WL) test (Shervashidze et al., 2011). Such works are useful for graph-level tasks such as graph classification or clustering, where the ability to differentiate subgraphs with different structures plays an essential role in performance. However, in many node-level tasks such as node classification, effective propagation of node-level information to distant neighbors in a graph is more important than having an expressive function for feature transformation. This leads to interesting directions of works: linear and decoupled GNNs.

TO DO...

Linear graph neural networks Linear GNNs have been studied widely in literature to improve the robustness and generalizability of GNNs. Wu et al. (2019) proposed simplified graph convolution (SGC) by removing the nonlinear activation functions from graph convolutional networks

[1] This problem definition should be changed based on the experimental setting.

Table 1: A summary of GNNs in the framework of LinGNN (in Definition 2). The * symbol in the name of a model represents that it is linearized by removing the nonlinear components (details are given in Section 3.2). Explain the concatenation operator.

Model	Type	$\mathcal{P}(\mathbf{A}, \mathbf{X})$
LR	Linear	$\mathbf{X}$
SGC	Linear	$\tilde{\mathbf{A}}_{\text{sym}}^{K}\mathbf{X}$
DGC	Linear	$[(1-T/K)\mathbf{I} + (T/K)\tilde{\mathbf{A}}_{\text{sym}}]^{K}\mathbf{X}$
S^2GC	Linear	$\sum_{k=1}^{K}(\alpha\mathbf{I} + (1-\alpha)\tilde{\mathbf{A}}_{\text{sym}}^{k})\mathbf{X}$
G^2CN	Linear	$\Vert_{i=1}^{N}[\mathbf{I} - (T_i/K)((b_i-1)\mathbf{I} + \mathbf{A}_{\text{sym}})^2]^{K}\mathbf{X}$
APPNP*	Decoupled	$[\sum_{k=0}^{K-1}\alpha(1-\alpha)^k\tilde{\mathbf{A}}_{\text{sym}}^{k} + (1-\alpha)^K\tilde{\mathbf{A}}_{\text{sym}}^{K}]\mathbf{X}$
GPR-GNN*	Decoupled	$\Vert_{k=0}^{K}\tilde{\mathbf{A}}_{\text{sym}}^{k}\mathbf{X}$
GCN*	Linearized	Same as SGC
SAGE*	Linearized	$\Vert_{k=0}^{K}\mathbf{A}_{\text{row}}^{k}\mathbf{X}$
GIN*	Linearized	$((1+\epsilon)\mathbf{I} + \mathbf{A})^{K}\mathbf{X}$
GCNII*	Linearized	$\Vert_{k=0}^{K-2}\tilde{\mathbf{A}}_{\text{sym}}^{k}\mathbf{X} \Vert ((1-\alpha)\tilde{\mathbf{A}}_{\text{sym}}^{K} + \alpha\tilde{\mathbf{A}}_{\text{sym}}^{K-1})\mathbf{X}$

(GCN). They showed that SGC performs incredibly well in real-world graphs even with its linear structure. Wang et al. (2021) proposed Decoupled Graph Convolution (DGC), which improves SGC by changing the adjacency matrix with a temperature hyperparameter that controls the weight of the self-loops. Li et al. (2022) proposed Graph Gaussian Convolutional Networks (G^2CN), which is based on the result of detailed spectral analysis called concentration analysis.

Decoupled graph neural networks Decoupled GNNs separate the propagation of information through the graph structure from the update of hidden representations, addressing the limitations of existing GNNs such as oversmoothing or the difficulty of choosing the number of layers. Decoupled GNNs first learn the representations of nodes by node-wise models such as multilayer perceptrons (MLP) and then propagate the representations by propagation algorithms such as personalized random walks (Klicpera et al., 2019a;b; Chien et al., 2021). Such existing works assume the nonlinearity of operations, but their decoupled nature allows us to easily linearize their operations and get the linear versions, such as by replacing MLPs with a single linear layer.

3 Linearization of Graph Neural Networks

We formally define a linear graph neural network (GNN) in the framework of logistic regression and perform systematic linearization of existing GNN models, presenting notable observations.

3.1 Definition of Linear Graph Neural Networks

Definition 1 (LR) *Given a feature $\mathbf{X} \in \mathbb{R}^{n\times m}$ and a label $\mathbf{y} \in \mathbb{R}^{l}$, where $l \leq n$ is the number of observations, let $\mathbf{Y} \in \mathbb{R}^{l\times c}$ be the one-hot representation of $\mathbf{y}$, and y_{ij} be the (i,j)-th element in $\mathbf{Y}$. Then, logistic regression (LR) is to find an optimal weight matrix $\mathbf{W}^* \in \mathbb{R}^{m\times c}$ as follows:*

$$\text{LR}(\mathbf{X}, \mathbf{y}) = \arg\max_{\mathbf{W}} \sum_{i=1}^{l}\sum_{j=1}^{c} y_{ij}\log\hat{y}_{ij} \;\text{ where }\; \hat{y}_{ij} = \frac{\exp(\mathbf{w}_{\cdot j}^{\top}\mathbf{x}_i)}{\sum_k \exp(\mathbf{w}_{\cdot k}^{\top}\mathbf{x}_i)}, \tag{1}$$

and $\mathbf{w}_{\cdot j}$ is the j-th column of $\mathbf{W}$. We omit the bias term for brevity without loss of generality.

Definition 2 (LinGNN) *Given a graph $G = (\mathbf{A}, \mathbf{X})$, let f be a node classifier function that predicts the labels of all nodes in G as $\hat{\mathbf{y}} = f(\mathbf{A}, \mathbf{X}; \Theta)$, where Θ is the set of learnable parameters. Then, f is* LinGNN *if $\Theta = \{\mathbf{W}\}$ and an optimal weight matrix $\mathbf{W}^*$ is defined as*

$$\mathbf{W}^* = \text{LR}(\mathcal{P}(\mathbf{A}, \mathbf{X}), \mathbf{y}), \tag{2}$$

where $\mathcal{P}$ is a feature propagator *function that determines the characteristic of f without learnable parameters. We ignore the bias term for brevity without loss of generality.*

We define LinGNN based on logistic regression with the function $\mathcal{P}$ that propagates given features through a graph. The only nonlinearity is the softmax function, which is necessary for classification tasks and does not harm the global interpretability of the weight matrix $\mathbf{W}$.[2] We represent existing linear models in the framework of LinGNN as follows:

Example 1 *SGC (Wu et al., 2019) is LinGNN with* $\mathcal{P}(\mathbf{A}, \mathbf{X}) = \tilde{\mathbf{A}}_{\text{sym}}^{K}\mathbf{X}$*, where* $K \geq 0$ *is a hyperparameter.*

Example 2 *DGC (Wang et al., 2021) is LinGNN with* $\mathcal{P}(\mathbf{A}, \mathbf{X}) = [(1-T/K)\mathbf{I}+(T/K)\tilde{\mathbf{A}}_{\text{sym}}]^{K}\mathbf{X}$*, where* $T \geq 0$ *and* $K \geq 0$ *are hyperparameters.*

Example 3 *G^{2}CN (Li et al., 2022) is LinGNN with* $\mathcal{P}(\mathbf{A}, \mathbf{X}) = \|_{i=1}^{N}[\mathbf{I} - (T_i/K)((b_i - 1)\mathbf{I} + \mathbf{A}_{\text{sym}})^2]^{K}\mathbf{X}$*, where* N*,* T_i*,* b_i*, and* $K \geq 0$ *are hyperparameters.*

G^{2}CN is the latest linear model that shows the highest accuracy on real-world graphs. If we compare the propagator functions of those models, it is clear that the complexity of the propagator function $\mathcal{P}$ increases sharply with involving more hyperparameters. That is, the improvement of accuracy in recent models is the result of sacrificing simplicity and interpretability of decisions, since complex feature transformation makes it difficult to interpret the learned weights. Considering that one of the most notable advantages of linear models is the global interpretability from the learned weights, this can be seen as losing the core value of choosing the linear structure of GNNs.

One motivation of this work is to design a simple and interpretable linear GNN whose feature transformation is corresponding with human intuition. We introduce our proposed approach in Section 5. Before that, we discuss more on the linearization of existing non-linear models in Section 3.2.

3.2 Linearization of Existing Models

The concept of linear GNNs in Definition 2 allows us to understand the core mechanism of different GNNs regarding feature propagation. The idea is applied not just to already linear GNNs, but also to nonlinear models if accompanied with a *linearization* process. We define the linearization of GNNs as Definition 3 and represent existing GNNs in the framework of LinGNN in Table 1.

Definition 3 (Linearization) *The linearization of a GNN is to select its hyperparameters in a way to remove all nonlinear components, representing it as LinGNN of Definition 2.*

Linear models SGC (Wu et al., 2019), DGC (Wang et al., 2021), and G^{2}CN (Li et al., 2022) are linear models whose propagator functions are based on the power of the adjacency matrix normalized symmetrically. DGC improves SGC by introducing a new hyperparameter T, which allows it to have a large number K of steps by adjusting the weight of self-loops in the adjacency matrix, and then G^{2}CN improves DGC by introducing more hyperparameters such as N and b_i. The process of representing the propagator function of G^{2}CN as in Table 1 is given in Appendix A.1.

Decoupled models PPNP, APPNP (Klicpera et al., 2019a), and GPR-GNN (Chien et al., 2021) are decoupled GNNs that use the personalized PageRank to propagate node representations. PPNP aims to get the exact solution of the random walk, which involves the expensive inverse operation on the adjacency matrix, while APPNP improves its efficiency with approximation. GPR-GNN further generalizes APPNP by introducing a learnable parameter for each k-hop propagated feature so that the importance of k-hop neighborhood is learned as a result of training.

The linearization of such decoupled GNNs is straightforward due to the decoupling between representation learning and propagation; we substitute a linear projection for a multilayer perceptron and run their propagation functions. Still, our linearization makes a more compact representation of the propagator functions, which allows us to understand their underlying mechanism. The linearization processes of APPNP and GPR-GNN are given in Appendix A.2 and A.3, respectively.

Linearizable models We linearize popular GNNs such as GCN (Kipf & Welling, 2017), GraphSAGE (Hamilton et al., 2017), GIN (Xu et al., 2019), and GCNII (Chen et al., 2020). The linearized version of GCN is the same as SGC, and the linearized version of GraphSAGE can be understood

[2] Definition 2 can be extended to regression tasks by replacing logistic regression with linear regression.

as combining features propagated with different hops as in G^2CN. The linearized version of GIN is different from SGC as it a) does not normalize $\mathbf{A}$ and b) gives a larger weight to the identity matrix $\mathbf{I}$ with a hyperparameter ϵ. GCNII is similar to GraphSAGE in that features propagated with different hops are concatenated in the propagator function. The difference is that the two terms with the most propagation steps are combined with different weights, determined by a hyperparameter α. Refer to Appendix A.4, A.5, and A.6 for the lineraizations of GraphSAGE, GIN, and GCNII, respectively.

Non-linearizable models Not every GNN is easily linearizable to our LinGNN framework, which requires the propagator function $\mathcal{P}$ to contain no learnable parameters. One category of GNNs whose linearization is difficult is attention-based models (Velickovic et al., 2018; Kim & Oh, 2021; Brody et al., 2022), which determine the adjacency matrix independently for each layer based on learnable parameters and mostly with the softmax function. It is possible to think of a linear variant that works in a similar way in terms of learning dynamic weights between every pair of adjacent nodes, but it is difficult to derive an exact linear variant in a systematic way. Thus, we leave it as a future work, but we still provide a process of incomplete linearization of GAT in Appendix B.1.

3.3 Observations from the Linearized Models

The result of linearization shown in Table 1 presents notable observations (or insights) that help one design a linear model that gains in both accuracy and interpretability.

Feature concatenation Table 1 shows that many GNNs improve their expressiveness (and performance) by concatenating the features propagated for different numbers of steps. This is an effective strategy for linear models due to the following two reasons: a) the complexity of a decision is limited due to the linearity, and b) a large number of parameters are not difficult to train.

Interpretability of propagator functions The linearized models in Table 1 have different degrees of interpretability based on the complexity of the propagator functions, although the decisions are all linear with the given feature $\mathbf{X}$. Models with simple propagator functions, such as SGC, GPR-GNN, and GraphSAGE, support the direct interpretability of the weight matrix $\mathbf{W}$, since the features make a direct relationship with predictions with a normalized adjacency matrix. On the other hand, models with complex propagator functions, such as DGC, G^2CN, APPNP, and GCNII, make it difficult to interpret the weight matrix even with the linearity of decisions.

Manipulation of the adjacency matrix Different GNN models in Table 1 adopt different strategies to normalize (or improve) the adjacency matrix $\mathbf{A}$.

4 Unit Tests for All Possible Graph Scenarios

The main goal of this research is to devise a universal classifier that works on all possible scenarios of real-world graphs in terms of the relationship between features, labels, and a structure. We formally define the possible scenarios of real-world graphs for node classification, summarize them based on the mutual information between variables, and perform experiments of existing GNNs.

4.1 Possible Graph Scenarios

We categorize all possible scenarios of node classification based on the characteristics of node features $\mathbf{X}$, a graph structure $\mathbf{A}$, and node labels $\mathbf{y}$. We denote random variables representing the edge (i, j) between nodes i and j and the label y_i of node i by A_{ij} and Y_i, respectively.

Edges For an adjacency matrix $\mathbf{A}$, we consider the following three cases:

- **Uniform:** $P(A_{ki} = 1 \mid A_{ij} = A_{jk} = 1) = P(A_{ki} = 1)$
- **Clustered:** $P(A_{ki} = 1 \mid A_{ij} = A_{jk} = 1) > P(A_{ki} = 1)$
- **Bipartite:** $P(A_{ki} = 1 \mid A_{ij} = A_{jk} = 1) < P(A_{ki} = 1)$

The uniform case represents that every edge is determined independently of the others, and the graph structure gives no useful information for node classification. In the clustered case, it is more likely that nodes having common neighbors make more connections in a graph. In the bipartite case, nodes make more connections with those sharing no common neighbors with them.

Table 2: Nine possible cases of graph information, grouped by the mutual information (MI) between features $\mathbf{X}$, edges $\mathbf{A}$, and labels $\mathbf{y}$. In the first case, the graph structure provides noisy information, because labels are independent of the structure. In the second case, the features do not provide useful information. In the last case, all three variables are related to each other.

MI	Features $\mathbf{X}$	Edges $\mathbf{A}$	Labels $\mathbf{y}$
$\mathbf{X}, \mathbf{y}$	Semantic	Uniform, Clustered, or Bipartite	Individual
$\mathbf{A}, \mathbf{y}$	Random	Clustered	Homophily
$\mathbf{A}, \mathbf{y}$	Random	Bipartite	Heterophily
$\mathbf{X}, \mathbf{A}, \mathbf{y}$	Structural or Semantic	Clustered	Homophily
$\mathbf{X}, \mathbf{A}, \mathbf{y}$	Structural or Semantic	Bipartite	Heterophily

Labels For the labels in $\mathbf{y}$, which are the target of node classification, we consider the following:

- **Individual:** $P(Y_i = y \mid A_{ij} = 1, Y_j = y) = P(Y_i = y)$
- **Homophily:** $P(Y_i = y \mid A_{ij} = 1, Y_j = y) > P(Y_i = y)$
- **Heterophily:** $P(Y_i = y \mid A_{ij} = 1, Y_j = y) < P(Y_i = y)$

The individual case means that the label of a node is independent of the labels of its neighbors. This is the case when the graph structure works as noise information with regard to classification. In the homophily case, adjacent nodes are likely to have the same label. It is the most popular assumption of GNNs for node classification. In the heterophily case, adjacent nodes are likely to have different labels, which is not as common as homophily but observed often in real-world graphs.

Features For a feature matrix $\mathbf{X}$, we consider the following three cases based on $\mathbf{A}$ and $\mathbf{y}$:

- **Random:** $p(\mathbf{x}_i, \mathbf{x}_j \mid y_i, y_j, a_{ij}) = p(\mathbf{x}_i, \mathbf{x}_j)$
- **Structural:** $p(\mathbf{x}_i, \mathbf{x}_j \mid y_i, y_j, a_{ij}) \neq p(\mathbf{x}_i, \mathbf{x}_j \mid y_i, y_j)$
- **Semantic:** $p(\mathbf{x}_i, \mathbf{x}_j \mid y_i, y_j, a_{ij}) \neq p(\mathbf{x}_i, \mathbf{x}_j \mid a_{ij})$

The random case represents that features are determined independently of the graph structure and labels. In this case, node features do not provide meaningful information for classification, but can work as unique indices of nodes like one-hot embeddings when the dimensionality m of features is large. In the structural case, features are related with the graph structure, but not directly with labels. They provide useful information for classification only if the graph structure and labels are related. In the semantic case, features directly provide useful information for predicting labels.

Feasibility Although there exist a total of 27 combinations for $(\mathbf{X}, \mathbf{A}, \mathbf{y})$, not all of them are possible to implement; for example, either homophily or heterophily $\mathbf{y}$ is not compatible with uniform $\mathbf{A}$, since the degree of homophily (or heterophily) is severely limited with the uniform distribution of edges. In summary, the following choices of variables are not compatible to each other:

- { Uniform, Clustered } $\mathbf{A}$ + Heterophily $\mathbf{y}$
- { Uniform, Bipartite } $\mathbf{A}$ + Homophily $\mathbf{y}$
- Structural $\mathbf{X}$ + Uniform $\mathbf{A}$

After removing the infeasible combinations of variables, we categorize the remaining ones based on the mutual information (MI) between variables in Table 2. For example, $\mathbf{X}$ and $\mathbf{y}$ have MI if $\mathbf{X}$ is semantic, regardless of the type of $\mathbf{A}$ or $\mathbf{y}$. We safely ignore the cases where $\mathbf{y}$ has no MI with any of the other variables, since every classifier is expected to fail in such cases.

4.2 Implementation of Graph Scenarios

There are various ways to generate synthetic graphs satisfying the definitions of different scenarios. One can use a synthetic graph generator designed to create more plausible graphs (Leskovec et al., 2010; Barabási & Albert, 1999), but we choose the simplest one to focus on the mutual information, rather than the other characteristics of synthetic graphs such as the degree distribution.

Structure We assume that the number of structural clusters is the same as the number c of labels for the alignment with label information. We divide all nodes into c groups and then decide the edge

densities for intra- and inter-connections of groups based on the structural type: uniform, clustered, and bipartite. We use a hyperparameter ϵ_a to determine the noise level in the case of homophily or heterophily: for example, if $\epsilon_a = 0$, the graph has a full block-diagonal adjacency matrix in the case of homophily. The expected number of edges is the same for all three cases.

A notable characteristic of a bipartite structure $\mathbf{A}$ is that $\mathbf{A}^2$ has a clustered structure. If we create inter-group connections for every possible pair of different groups, even noise-free $\mathbf{A}$ with $\epsilon_a = 0$ makes noisy $\mathbf{A}^2$ with inter-group connections. For better consistency, we set the number of classes to an even number in our experiments, randomly pick paired classes such as $(1, 3)$ and $(2, 4)$ when $c = 4$, for example, and create inter-group connections only for the chosen pairs. In this way, we create a non-diagonal block-permutation matrix $\mathbf{A}$ when $\epsilon_a = 0$, and the noise level of $\mathbf{A}^2$, which has a clustered structure, is solely controlled by the noise level of $\mathbf{A}$.

Labels In the case of individual $\mathbf{y}$, we determine the label of each node uniformly at random, with no consideration of the graph structure. In the case of homophily or heterophily $\mathbf{y}$, we assign labels based on the groups of nodes assumed by the graph structure. That is, nodes in the same group have the same label, and thus the group index itself works as the label $\mathbf{y}$. In this way, we force homophily $\mathbf{y}$ for clustered $\mathbf{A}$, and heterophily $\mathbf{y}$ for bipartite $\mathbf{A}$. The degree of homophily (or heterophily) is also determined by the noise level ϵ_a of the graph structure.

Features We basically assume that every feature element has zero-mean and unit-variance. Thus, in the random case, we sample each element from the standard normal distribution $\mathcal{U}(0, 1)$. In the structural case, we run low-rank support vector decomposition (SVD) (Halko et al., 2011) to make $\mathbf{X}$ have structural information. Given $\mathbf{U}\Sigma\mathbf{V}^\top \approx \mathbf{A}$ from low-rank SVD, we take $\mathbf{U}$ and normalize each feature element to have the zero-mean and unit-variance. The rank r in the SVD is determined as a hyperparameter; higher r captures the structure better, but can give noisy information.

In the semantic case, we randomly pick c representative vectors $\{\mathbf{v}_k\}_{k=1}^c$ in the unit hypersphere, which correspond to the c different classes. Then, for each node i with label y, we sample a vector $\mathbf{x}$ from $\mathcal{U}(0, 1)$ such that $\arg\max_k \mathbf{x}^\top \mathbf{v}_k = y$. In this way, we have random Gaussian vectors having sufficient semantic information for the classification of labels, with a guarantee that the perfect linear decision boundaries can be drawn in the feature space $\mathbf{X}$ at the training time.

4.3 Unit Tests on Synthetic Graphs

We run unit tests on synthetic graphs to check which GNNs fail on each type of graphs. We include the following baselines: LR (logistic regression), SGC (Wu et al., 2019), and the linearized version of GraphSAGE (Hamilton et al., 2017). We set the number of layers in all GNNs to 2.

Table 3 shows the results of unit tests. LR works well when $\mathrm{MI}(\mathbf{X}, \mathbf{y}) > 0$, since it learns the direct relationship between features and labels. Still, the accuracy of LR on structural $\mathbf{X}$ is not as high as on semantic $\mathbf{X}$, since the structural features provide only partial information on the graph structure, which is related to the labels. SGC and SAGE* improve the accuracy of LR when $\mathrm{MI}(\mathbf{A}, \mathbf{y}) > 0$ by propagating observed information through the graph, but decreases the accuracy when $\mathbf{A}$ is noisy. Our proposed approach, on the other hand, shows high accuracy in all types of synthetic graphs. We provide details on our proposed approach and how it successes in Section 5.

5 Proposed Method

In designing a novel linear GNN that can address all unit tests effectively, the following questions should be addressed:

1. How can we choose only the meaningful features for each dataset or task?

2. How can we properly normalize the adjacency matrix to deal with multiple datasets?

Table 3: The classification accuracies of linear models on various types of synthetic graphs. Existing models fail on at least one type of graphs, but our proposed approach works well in all scenarios, proving its robustness and generalizability. Details are given in Section 4.3.

MI	X	A	y	LR	SGC	SAGE*	Ours
$\mathbf{X}, \mathbf{y}$	Semantic	Uniform	Individual	96.5	58.3	90.4	90.4
$\mathbf{X}, \mathbf{y}$	Semantic	Clustered	Individual	96.5	60.8	91.1	90.9
$\mathbf{X}, \mathbf{y}$	Semantic	Bipartite	Individual	96.5	57.1	88.4	89.3
$\mathbf{A}, \mathbf{y}$	Random	Clustered	Homophily	24.6	43.7	36.3	70.2
$\mathbf{A}, \mathbf{y}$	Random	Bipartite	Heterophily	24.6	27.8	37.7	69.2
$\mathbf{X}, \mathbf{A}, \mathbf{y}$	Structural	Clustered	Homophily	73.2	86.3	82.8	82.8
$\mathbf{X}, \mathbf{A}, \mathbf{y}$	Structural	Bipartite	Heterophily	73.0	70.9	85.0	84.0
$\mathbf{X}, \mathbf{A}, \mathbf{y}$	Semantic	Clustered	Homophily	95.3	96.8	97.4	97.5
$\mathbf{X}, \mathbf{A}, \mathbf{y}$	Semantic	Bipartite	Heterophily	95.3	76.7	97.0	97.3

5.1 Proper Normalization for GraphSAGE

Let $\mathbf{D}^{(K)}$ be the diagonal degree matrix of $\mathbf{A}^K$, where $d_{ii}^{(K)} = \sum_j a_{ij}^K$. Then, we have:

$$d_{ii}^{(K)} = \sum_j (\mathbf{A}^K)_{ij} \tag{3}$$

$$= \sum_j \sum_k a_{ik} (\mathbf{A}^{K-1})_{kj} \tag{4}$$

$$= \sum_k a_{ik} \sum_j (\mathbf{A}^{K-1})_{kj} \tag{5}$$

$$= \sum_k a_{ik} d_{kk}^{(K-1)} \tag{6}$$

$$= \sum_k (\mathbf{A}\mathbf{D}^{(K-1)})_{ik} \tag{7}$$

We change it into the vector form as follows:

$$\mathbf{D}^{(K)} = \mathrm{diag}(\mathrm{rowsum}(\mathbf{A}\mathbf{D}^{(K-1)})), \tag{8}$$

where $\mathrm{diag}(\cdot)$ is a function that makes a diagonal matrix from a vector, and $\mathrm{rowsum}(\cdot)$ is a function that computes the sums of rows of a matrix. The computation of Equation equation 8 is $O(|\mathcal{E}|)$.

6 Experiments

We present experimental results on real-world datasets containing both homophily and heterophily graphs.

6.1 Experimental Settings

Datasets Since the heterophily graphs are directed, we perform the preprocessing as in an existing work (Chien et al., 2021) to make them undirected.

7 Conclusion

In this work, we present a systematic, comprehensive study on linear graph neural networks (GNN). First, we categorize possible types of real-world graphs based on the mutual information of variables and show that none of the existing GNNs successes to work well in all settings. Second, we formally define a linear GNN and present the result of linearization of existing models to compare their ideas of propagation without involving nonlinear operations. Third, we propose our approach that works well in every unit test.

REFERENCES

Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286 (5439):509–512, 1999.

Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? In *ICLR*, 2022.

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *ICML*, 2020.

Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. In *ICLR*, 2021.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, 2016.

Vikas K. Garg, Stefanie Jegelka, and Tommi S. Jaakkola. Generalization and representational limits of graph neural networks. In *ICML*, 2020.

Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Rev.*, 53(2): 217–288, 2011.

William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, 2017.

Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin R. Benson. Combining label propagation and simple models out-performs graph neural networks. In *ICLR*, 2021.

Dongkwan Kim and Alice Oh. How to find your friendly neighborhood: Graph attention design with self-supervision. In *ICLR*, 2021.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *ICLR*, 2019a.

Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. In *NeurIPS*, 2019b.

Jure Leskovec, Deepayan Chakrabarti, Jon M. Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. Kronecker graphs: An approach to modeling networks. *J. Mach. Learn. Res.*, 11:985–1042, 2010.

Guohao Li, Matthias Müller, Ali K. Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *ICCV*, 2019.

Mingjie Li, Xiaojun Guo, Yifei Wang, Yisen Wang, and Zhouchen Lin. G^2cn: Graph gaussian convolution networks with concentrated graph filters. In *ICML*, 2022.

Pál András Papp, Karolis Martinkus, Lukas Faber, and Roger Wattenhofer. Dropgnn: Random dropouts increase the expressiveness of graph neural networks. In *NeurIPS*, 2021.

Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *J. Mach. Learn. Res.*, 12:2539–2561, 2011.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

Yifei Wang, Yisen Wang, Jiansheng Yang, and Zhouchen Lin. Dissecting the diffusion process in linear graph convolutional networks. *NeurIPS*, 2021.

Felix Wu, Amauri H. Souza Jr., Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying graph convolutional networks. In *ICML*, 2019.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Trans. Neural Networks Learn. Syst.*, 32 (1):4–24, 2021.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *KDD*, 2018.

Jaemin Yoo, Hyunsik Jeon, and U Kang. Belief propagation network for hard inductive semi-supervised learning. In *IJCAI*, 2019.

Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

Hao Zhu and Piotr Koniusz. Simple spectral graph convolution. In *ICLR*, 2021.

A LINEARIZATION PROCESSES

To move this into the main paper?

We present detailed processes of linearizing graph neural networks (GNN). Refer to Section 2 of the main paper for frequently used symbols. Linearization is done in two steps. First, we replace all nonlinear components such as activation functions from the original layer definition of each GNN. Second, if the resulting layer function contains learnable parameters, we devise a replacement that is at least as expressive as the given function while having no learnable parameters. This is because our goal of linearization is not just representing a linear decision function, but understanding GNNs in the framework of logistic regression based on our definition of LinGNN in Definition 2. We use the symbol $\preceq$ (or $\succeq$) to represent the "at least as expressive as" relation.

A.1 G^{2}CN (LI ET AL., 2022)

Li et al. (2022) propose various formulations of Graph Gaussian Convolutional Networks (G^{2}CN). The G^{2}CN-Euler model, which is the main model in their experiments, is given as follows:

$$\mathcal{P}(\mathbf{A}, \mathbf{X}; \{\theta_i\}_{i=1}^N) = \sum_{i=1}^N \theta_i \mathbf{H}_{i,t}, \tag{9}$$

where θ_i is a learnable parameter, the transformed feature $\mathbf{H}_{i,t}$ is defined as

$$\begin{aligned}
\mathbf{H}_{i,t} &= \mathbf{H}_{i,t-1} - \frac{T_i}{K}(\mathbf{L} - b_i\mathbf{I})^2\mathbf{H}_{i,t-1} \\
&= \mathbf{H}_{i,t-1} - \frac{T_i}{K}((b_i - 1)\mathbf{I} + \mathbf{A}_{\text{sym}})^2\mathbf{H}_{i,t-1} \\
&= [\mathbf{I} - \frac{T_i}{K}((b_i - 1)\mathbf{I} + \mathbf{A}_{\text{sym}})^2]\mathbf{H}_{i,t-1} \\
\mathbf{H}_{i,0} &= \mathbf{X},
\end{aligned} \tag{10}$$

and N, T_i, and b_i are hyperparameters. The original paper (Li et al., 2022) choose the hyperparameters $N = 2$, $b_1 = 0$, $b_2 = 2$, and $T_i \geq 0$ by hyperparameter tuning for each dataset.

Since the transformed features are linearly combined with a learnable parameter θ_i in Equation 9, we make a propagator function $\mathcal{P}' \succeq \mathcal{P}$ that contains no learnable parameters as follows:

$$\mathcal{P}'(\mathbf{A}, \mathbf{X}) = \mathop{\Big\|}_{i=1}^{N} \mathbf{H}_{i,t}. \tag{11}$$

The representation of Equation 9 can be considered as a squeezed version of Equation 11, which reduces the number of parameters by sharing the weight matrix between different transformed features only with a difference of a scalar multiplication by θ_i. Still, Equation 11 allows us to understand the decision mechanism of the model in the framework of logistic regression.

A.2 APPNP (KLICPERA ET AL., 2019A)

We assume that the initial node representation is created by a single linear layer of $\mathbf{XW}$, where $\mathbf{W}$ is a weight matrix. Then, the t-th representation matrix $\mathbf{H}_t$ is represented as follows:

$$\mathbf{H}_t = (1 - \alpha)\tilde{\mathbf{A}}_{\text{sym}}\mathbf{H}_{t-1} + \alpha\mathbf{XW}, \tag{12}$$

where $0 < \alpha < 1$ is a hyperparameter. If we apply a chain of two layers for generalization, we get the following:

$$\mathbf{H}_t = [(1 - \alpha)\tilde{\mathbf{A}}_{\text{sym}}]^2\mathbf{H}_{t-2} + \alpha(1 - \alpha)\tilde{\mathbf{A}}_{\text{sym}}\mathbf{XW} + \alpha\mathbf{XW}. \tag{13}$$

If we generalize it into K layers, we get the following:

$$\mathbf{H}_K = \left([(1 - \alpha)\tilde{\mathbf{A}}_{\text{sym}}]^K + \alpha\sum_{k=0}^{K-1}[(1 - \alpha)\tilde{\mathbf{A}}_{\text{sym}}]^k\right)\mathbf{XW}. \tag{14}$$

We safely remove the weight matrix $\mathbf{W}$, which is redundant, and get the representation in Table 1.

A.3 GPR-GNN (Chien et al., 2021)

GPR-GNN is a decoupled GNN like APPNP. If we assume the initial node representation of $\mathbf{XW}$ as in Appendix A.2, where $\mathbf{W}$ is a weight matrix, we have the following propagator function:

$$\mathcal{F}(\mathbf{A}, \mathbf{X}; \{\theta_k\}_{k=0}^{K}) = \sum_{k=0}^{K} \theta_k \tilde{\mathbf{A}}_{\text{sym}}^{k} \mathbf{XW}, \tag{15}$$

where θ_k is a parameter that is learned together with $\mathbf{W}$. We safely remove $\mathbf{W}$ as in Appendix A.2, because it is redundant due to the weight matrix learned in the LR framework. We then replace the summation term with concatenation to remove the learnable parameters from $\mathcal{F}$ as in Appendix A.1. Then, we have the representation in Table 1.

A.4 GraphSAGE (Hamilton et al., 2017)

We assume the mean aggregator of GraphSAGE. By replacing the activation function as the identity function, each layer $\mathcal{F}$ of GraphSAGE is linearized as follows:

$$\mathcal{F}(\mathbf{X}) = \mathbf{XW}_1 + \mathbf{A}_{\text{row}}\mathbf{XW}_2, \tag{16}$$

where $\mathbf{A}_{\text{row}} = \mathbf{D}^{-1}\mathbf{A}$ represents the mean operator in the aggregation, and $\mathbf{W}_1$ and $\mathbf{W}_2$ are learnable weight matrices in the layer. If we apply a chain of two layers, where the weight matrices of the second layer are represented as $\mathbf{W}_3$ and $\mathbf{W}_4$, we get the following:

$$\mathcal{F}(\mathcal{F}(\mathbf{X})) = \mathbf{XW}_1\mathbf{W}_3 + \mathbf{A}_{\text{row}}\mathbf{X}(\mathbf{W}_1\mathbf{W}_2 + \mathbf{W}_2\mathbf{W}_3) + \mathbf{A}_{\text{row}}^2\mathbf{XW}_2\mathbf{W}_4 \tag{17}$$

$$= \mathbf{XW}_a + \mathbf{A}_{\text{row}}\mathbf{XW}_b + \mathbf{A}_{\text{row}}^2\mathbf{XW}_c, \tag{18}$$

where we redefine the weight matrices without loss of generality as

$$\mathbf{W}_a = \mathbf{W}_1\mathbf{W}_3 \tag{19}$$

$$\mathbf{W}_b = \mathbf{W}_1\mathbf{W}_2 + \mathbf{W}_2\mathbf{W}_3 \tag{20}$$

$$\mathbf{W}_c = \mathbf{W}_2\mathbf{W}_4. \tag{21}$$

If we generalize it into K layers, we get the following:

$$\mathcal{F}^K(\mathbf{X}) = \sum_{k=1}^{K} \mathbf{A}_{\text{row}}^{k}\mathbf{XW}_k. \tag{22}$$

Note that a different weight matrix $\mathbf{W}_k$ is applied to each layer k. This is equivalent to concatenating the transformed features of all layers and learning a single large weight matrix in training.

A.5 GIN (Xu et al., 2019)

The linearization of GIN is straightforward, since it applies a multilayer perceptron (MLP) to each layer after aggregating the representations of adjacent nodes. After we replace the MLP with a linear layer, we rewrite each layer $\mathcal{F}$ of GIN as follows:

$$\mathcal{F}(\mathbf{X}) = ((1+\epsilon)\mathbf{I} + \mathbf{A})\mathbf{XW}, \tag{23}$$

where ϵ is either a constant or a learnable parameter, and $\mathbf{W}$ is a weight matrix. The original paper (Xu et al., 2019) presents two variants of GIN based on the type of ϵ. GIN-0 sets ϵ to zero, and has the following linearized layer:

$$\mathcal{F}_{\text{GIN-0}}(\mathbf{X}) = \tilde{\mathbf{A}}\mathbf{XW}, \tag{24}$$

where $\tilde{\mathbf{A}} = \mathbf{I} + \mathbf{A}$. This results in a similar propagator function as in SGC only with a difference of the normalization of $\mathbf{A}$. On the other hand, GIN-ϵ, which treats ϵ as a learnable parameter, has a more complex representation of the layer:

$$\mathcal{F}_{\text{GIN-}\epsilon}(\mathbf{X}) = \mathbf{XW}' + \tilde{\mathbf{A}}\mathbf{XW}, \tag{25}$$

where we generalize $\epsilon\mathbf{W}$ as another weight matrix $\mathbf{W}'$, which can be learned freely separately from the original weight matrix $\mathbf{W}$. It is noteworthy that the result is very similar to the linearized version of GraphSAGE, except with a difference on how to normalize $\mathbf{A}$.

A.6 GCNII (CHEN ET AL., 2020)

After replacing the activation function with the identity function, the l-th layer $\mathcal{F}_l$ of GCNII is given as follows:

$$\mathcal{F}_l(\mathbf{H}) = ((1-\alpha_l)\tilde{\mathbf{A}}_{\text{sym}}\mathbf{H} + \alpha_l\mathbf{X}))((1-\beta_l)\mathbf{I} + \beta_l\mathbf{W}_l), \tag{26}$$

where α_l and β_l are hyperparameters, and $\mathbf{W}_l$ is a weight matrix. The second term is equivalent to $\mathbf{W}_l$ regardless of the value of β_l, since $\mathbf{W}_l$ is a free parameter. We also set α_l to a constant α which is the same for every layer l, following the original paper (Chen et al., 2020).[3] Then, the equation is simplified as

$$\mathcal{F}_l(\mathbf{H}) = ((1-\alpha)\tilde{\mathbf{A}}_{\text{sym}}\mathbf{H} + \alpha\mathbf{X})\mathbf{W}_l. \tag{27}$$

If we apply a chain of two layers l and $l+1$, we get the following:

$$\mathcal{F}_{l+1}(\mathcal{F}_l(\mathbf{H})) = ((1-\alpha)\tilde{\mathbf{A}}_{\text{sym}}((1-\alpha)\tilde{\mathbf{A}}_{\text{sym}}\mathbf{H} + \alpha\mathbf{X})\mathbf{W}_l + \alpha\mathbf{X})\mathbf{W}_{l+1} \tag{28}$$

$$= (1-\alpha)^2\tilde{\mathbf{A}}^2_{\text{sym}}\mathbf{H}\mathbf{W}_l\mathbf{W}_{l+1} + \alpha(1-\alpha)\tilde{\mathbf{A}}_{\text{sym}}\mathbf{X}\mathbf{W}_l\mathbf{W}_{l+1} + \alpha\mathbf{X}\mathbf{W}_{l+1} \tag{29}$$

$$= (1-\alpha)^2\tilde{\mathbf{A}}^2_{\text{sym}}\mathbf{H}\mathbf{W}'_l + \alpha(1-\alpha)\tilde{\mathbf{A}}_{\text{sym}}\mathbf{X}\mathbf{W}'_l + \alpha\mathbf{X}\mathbf{W}_{l+1}, \tag{30}$$

where $\mathbf{W}'_l = \mathbf{W}_l\mathbf{W}_{l+1}$, which is also a free parameter. If we generalize it into K layers, we get the following:

$$\mathcal{F}^K(\mathbf{X}) = (1-\alpha)^{K-1}((1-\alpha)\tilde{\mathbf{A}}^K_{\text{sym}} + \alpha\tilde{\mathbf{A}}^{K-1}_{\text{sym}})\mathbf{X}\mathbf{W}_K + \alpha\sum_{k=0}^{K-2}(1-\alpha)^k\tilde{\mathbf{A}}^k_{\text{sym}}\mathbf{X}\mathbf{W}_k. \tag{31}$$

We safely remove the constant from each term, which can be included in the weight matrix:

$$\mathcal{F}^K(\mathbf{X}) = ((1-\alpha)\tilde{\mathbf{A}}^K_{\text{sym}} + \alpha\tilde{\mathbf{A}}^{K-1}_{\text{sym}})\mathbf{X}\mathbf{W}_K + \sum_{k=0}^{K-2}\tilde{\mathbf{A}}^k_{\text{sym}}\mathbf{X}\mathbf{W}_k. \tag{32}$$

We replace the summation operators between terms having different weight matrices with concatenation operators, having the final propagator function $\mathcal{P}$ as follows:

$$\mathcal{P}(\mathbf{A}, \mathbf{X}) = \Big\|_{k=0}^{K-2}\tilde{\mathbf{A}}^k_{\text{sym}}\mathbf{X} \,\|\, ((1-\alpha)\tilde{\mathbf{A}}^K_{\text{sym}} + \alpha\tilde{\mathbf{A}}^{K-1}_{\text{sym}})\mathbf{X} \tag{33}$$

B INCOMPLETE LINEARIZATION

B.1 GAT

We linearize graph attention networks (GAT) Velickovic et al. (2018) with the following modifications. The main idea of GAT, which is to learn the elements of the adjacency matrix from the given features, is still preserved even with the modifications from the edge weight term e_{ij} from node j to node i.

1. We ignore the activation functions between layers.
2. We simplify the attention function $\alpha_{ij} = \exp(e_{ij})/\sum_k \exp(e_{ik})$ as e_{ij}.
3. We remove the LeaklyReLU function in the computation of e_{ij}.
4. We assume the single-head attention for simplicity.

The edge weight e_{ij}, which is the (i, j)-th element of the propagator matrix, is defined as follows:

$$e_{ij} = \mathbf{a}^\top_{\text{dst}}(\mathbf{W}^\top\mathbf{x}_i) + \mathbf{a}^\top_{\text{src}}(\mathbf{W}^\top\mathbf{x}_j), \tag{34}$$

where $\mathbf{x}_i$ and $\mathbf{x}_j$ are feature vectors of length m for node i and j, respectively, $\mathbf{W}$ is a $m \times c$ learnable weight matrix, and $\mathbf{a}_{\text{dst}}$ and $\mathbf{a}_{\text{src}}$ are learnable weight vectors of length c.

[3] α is set to 0.1 in the original paper of GCNII (Chen et al., 2020).

Based on Equation equation 34, we represent the initial form of a linearized layer $\mathcal{F}'$ of GAT as follows:

$$\mathcal{F}'(\mathbf{X}) = [\mathrm{diag}(\mathbf{X}\mathbf{W}\mathbf{a}_{\mathrm{dst}})\tilde{\mathbf{A}} + \tilde{\mathbf{A}}\mathrm{diag}(\mathbf{X}\mathbf{W}\mathbf{a}_{\mathrm{src}})]\mathbf{X}\mathbf{W}, \tag{35}$$

where $\mathbf{X}$ is a $n \times m$ feature matrix, and $\tilde{\mathbf{A}}$ is the adjacency matrix with self-loops.

Assuming that $\mathbf{W}$ is used independently in all three terms of Equation equation 35, we simplify the equation as follows by introducing $\max(0, 2(d-c))$ more parameters:

$$\mathcal{F}(\mathbf{X}) = [\mathrm{diag}(\mathbf{X}\mathbf{w}_{\mathrm{dst}})\tilde{\mathbf{A}} + \tilde{\mathbf{A}}\mathrm{diag}(\mathbf{X}\mathbf{w}_{\mathrm{src}})]\mathbf{X}\mathbf{W}, \tag{36}$$

where $\mathbf{w}_{\mathrm{dst}}$ and $\mathbf{w}_{\mathrm{src}}$ are learnable vectors of length m that replace $\mathbf{a}_{\mathrm{dst}}$ and $\mathbf{a}_{\mathrm{src}}$, respectively.