On the Initialization of Graph Neural Networks

Jiahang Li [1 *] **Yakun Song** [2 *] **Xiang Song** [3] **David Paul Wipf** [4]

Abstract

Graph Neural Networks (GNNs) have displayed considerable promise in graph representation learning across various applications. The core learning process requires the initialization of model weight matrices within each GNN layer, which is typically accomplished via classic initialization methods such as *Xavier* initialization. However, these methods were originally motivated to stabilize the variance of hidden embeddings and gradients across layers of Feedforward Neural Networks (FNNs) and Convolutional Neural Networks (CNNs) to avoid vanishing gradients and maintain steady information flow. In contrast, within the GNN context classical initializations disregard the impact of the input graph structure and message passing on variance. In this paper, we analyze the variance of forward and backward propagation across GNN layers and show that the variance instability of GNN initializations comes from the combined effect of the activation function, hidden dimension, graph structure and message passing. To better account for these influence factors, we propose a new initialization method for **V**ariance **I**nstability **R**eduction within **G**NN **O**ptimization (Virgo), which naturally tends to equate forward and backward variances across successive layers. We conduct comprehensive experiments on 15 datasets to show that Virgo can lead to superior model performance and more stable variance at initialization on node classification, link prediction and graph classification tasks.

*Work was done during an internship in Amazon Shanghai AI Lab [1]The Hong Kong Polytechnic University [2]Shanghai Jiao Tong University [3]Amazon AI [4]Amazon Shanghai AI Lab. Correspondence to: Jiahang Li <jiahangspongebob.li@connect.polyu.hk>, Xiang Song <xiangsx@amazon.com>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

1. Introduction

Graph Neural Networks (GNNs) (Hamilton et al., 2017; Veličković et al., 2017; Kipf & Welling, 2016; Xu et al., 2018a; Monti et al., 2017) are a class of deep learning models specifically designed to process and analyze graph-structured data, allowing for the integration of both node and edge information for tasks such as node classification, link prediction, and graph classification. GNNs have recently shown great success in graph representation learning in service of various downstream applications including social networks (Ying et al., 2018; Rossi et al., 2020), recommendation (Fan et al., 2019; Yu et al., 2021), fraud detection (Wang et al., 2019; Liu et al., 2020), and life sciences (Strokach et al., 2020; Jing & Xu, 2021; Nguyen et al., 2020).

To initiate model training, learnable GNN weight matrices need to be initialized in one way or another. In the past, more traditional deep learning models (e.g., CNNs, MLPs) have typically adopted initialization schemes designed to improve training outcomes by stabilizing the variance of forward and backward passes (LeCun et al., 2012; Glorot & Bengio, 2010; He et al., 2015), the motivation being that unstable variances could otherwise lead to undesirable phenomena such as vanishing gradients (LeCun et al., 2012) or poor information flow (Glorot & Bengio, 2010). The GNN community has largely borrowed these same schemes, particularly the *Xavier* (Glorot & Bengio, 2010) and *Lecun* (LeCun et al., 2012) initialization paradigms. For example, the Deep Graph Library (DGL) (Wang et al., 2020) uses *Xavier* to initialize the layers of GNN architectures such as GCN (Kipf & Welling, 2016), GraphSAGE (Hamilton et al., 2017), GAT/GATv2 (Veličković et al., 2017; Brody et al., 2021) and SGC (Wu et al., 2019) models. Similarly, the PyTorch Geometric (PyG) package (Fey & Lenssen, 2019) also uses *Xavier* to initialize models like GCN, GAT/GATv2, and RGCN (Schlichtkrull et al., 2018). Meanwhile, RGCN and ChebNet (Defferrard et al., 2016) layers within DGL and GraphSAGE layers within PyG adopt *Lecun* initialization.

And yet despite this widespread adoption within GNN training, it remains unclear the degree to which the original justifications for existing initializations actually still apply when we venture beyond the non-GNN architectures for which they were first designed. Indeed, prior analysis has largely relied on the assumption of i.i.d. training instances devoid of graph structure (and all neurons within each layer having the same variance), which greatly simplifies the selection of an appropriate distribution for drawing initial matrices. The latter is usually a zero-mean uniform or Gaussian distribution with variance chosen to reduce the influence of the model hidden dimension (LeCun et al., 2012; Glorot & Bengio, 2010; He et al., 2015) or activation function (He et al., 2015) on the variance. However, GNN message passing layers and graph structure can impact initial variances in more nuanced ways, for example, through dependencies introduced by varying node-wise receptive-field sizes. Hence prior assumptions may no longer apply and it behooves us to consider GNN-specific alternatives.

For this purpose, we first present derivations for forward and backward variances within a certain class of message passing GNNs. Specifically, for any given layer, we decompose the variance of each node into the sum of variances over message propagation paths, and then further decompose the variance of each message propagation path into the sum of variances over weight propagation paths. As a result of these cascaded decompositions, we obtain expressions for the variance of each node in terms of the variance of weight matrices. These expressions disclose the combined impact of hidden dimension, activation function, graph structure, and message aggregation mechanism of the variance of hidden embeddings and gradients.

Based on these insights, we next propose a simple but effective initialization method called *Virgo* for **V**ariance **I**nstability Reduction within **G**NN **O**ptimization to mitigate the influence of these factors. Defining the overall variance within each layer as the mean over the variances of all nodes, Virgo minimizes the difference of overall variance between successive layers, and thereby derives the variance of distributions, such as zero-mean Gaussian or uniform, from which we can sample initial weight matrices for GNNs. Finally, we conduct comprehensive experiments on 15 datasets across three popular graph tasks, namely, node classification, link prediction, and graph classification. We compare Virgo with existing initialization methods including *Lecun*, *Xavier* and *Kaiming*. Overall, we make following contributions:

- We derive and analyze expressions for the variance of GNN embeddings (forward pass) and gradients (backward pass), showcasing how these quantities are affected by the joint influences of hidden dimension, activation function, graph structure, and GNN message passing mechanisms.
- We propose a new initialization method named Virgo for GNN weight matrices based on our analysis, which minimizes the difference of the overall variance between successive layers.
- We evaluate GNNs with different initializations on node classification, link prediction and graph classification tasks. Virgo helps improve prediction accuracy by up to 7% and well stabilizes variances at the initialization.

2. Preliminaries

This section introduces basic GNN concepts and initialization methods. And for convenience, we summarize our adopted notational conventions in Table 1.

2.1. Graph Neural Networks

Given a graph $\mathcal{G} = (\boldsymbol{X}, \boldsymbol{A})$, where $\boldsymbol{X}$ is the feature matrix of nodes, $\boldsymbol{A}$ is the graph adjacency matrix, the forward propagation over $\mathcal{G}$ of the l-th GNN layer can be defined as:

$$\boldsymbol{h}_i^l = \sigma\left(\sum_{j\in\mathbb{N}(i)} d_{ij}\boldsymbol{h}_j^{l-1}\boldsymbol{W}^{l-1}\right). \tag{1}$$

In this expression, $\mathbb{N}(i)$ is a set including 1-hop neighbors of node i, σ is an activation function, which we assume to be ReLU in this paper, $\boldsymbol{W}^l \in \mathbb{R}^{m_1^{(l)}\times m_2^{(l)}}$ is a weight matrix, and $\boldsymbol{h}_i^l$ denotes the hidden embedding of the i-th node. We also set the scaling constant d_{ij} to $1/\sqrt{(d_i+1)(d_j+1)}$ following the GCN model (Kipf & Welling, 2016), where d_i and d_j are the degrees of node i and j respectively. We denote the t-th element of the hidden embedding $\boldsymbol{h}_i^l$ by $h_{i,t}^l$. $\bar{\boldsymbol{h}}_i^l$ and $var(h_i^l) = \mathrm{Var_t}(h_{i,t}^l)$ denote the mean and variance over neurons within $\boldsymbol{h}_i^l$ respectively. We use $var(h^l)$ to denote the mean over $var(h_i^l)$ of all nodes i. $var(h^l)$ is also termed as the **forward variance** of the l_{th} layer. Analogously, the mean over $var(\frac{\partial Loss}{\partial h_i^l})$ of all nodes i, denoted by $var(\frac{\partial Loss}{\partial h^l})$, is termed as the **backward variance** of the l_{th} layer, where $Loss$ refers to a standard cross entropy objective we assume throughout the paper. Hereinafter we use **variance** to denote these two types of variance without ambiguity.

At initialization, we assume that neurons within $\boldsymbol{h}_i^l$ are independent and identically distributed (i.i.d). The same assumption also applies to elements of $\boldsymbol{W}^l$. Analogously, we assume that for all neurons t, t', two certain nodes i, j and a layer l, $\frac{\partial h_{j,t'}^L}{\partial h_{i,t}^l}$ are i.i.d. We also assume that elements of $\boldsymbol{W}^l$ are independent to elements of $\boldsymbol{W}^{l'}$, where l is not equal to l'. The input features $\boldsymbol{h}_i^0$ are random variables and the following derivations are conditioned on the input features of the given graph $\mathcal{G}$.

2.2. Classic Initializations Used by GNNs

We denote the (i,j)-th of the weight matrix $\boldsymbol{W}^l$ by $w_{i,j}^l$. In the following, we remove subscripts (i,j) without caus-

Table 1. Notation table.

$\boldsymbol{W}^l$	Weight matrix of the l-th GNN layer, $\boldsymbol{W}^l \in \mathbb{R}^{m_1^{(l)} \times m_2^{(l)}}$. $\boldsymbol{W}_{ij}^l$ is denoted by $w_{i,j}^l$
$\hat{\boldsymbol{h}}_i^l$, $\boldsymbol{h}_i^l$	Embedding of node i at the l-th layer before and after the activation. $\boldsymbol{h}_i^0$ is input feature of node i
i, j	Node index
t	Neuron index
l, L	Layer index and number of layers of a GNN
$h_{i,t}^l$, w_{t_1,t_2}^l	The t-th element of embedding of node i at l-th layer, and (t_1, t_2)-th element of $\boldsymbol{W}^l$
d_{ij}	Re-normalization coefficient between node i and j following the definition of GCN (1)
$\mathbb{N}$, $\mathbb{N}(i)$	Set of nodes in $\mathcal{G}$ and one-hop neighbors of node i
$[\boldsymbol{h}_i^l]_p$	Message propagation path indexed by p. This path has length l and takes i as the destination node. The set of all such p is denoted as $\mathbb{P}_{i,l}$
$[h_i^l]_{p,t,\phi}$	Weight propagation path indexed by ϕ. This path goes along the message propagation path p, is connected to the t-th neuron of node i and has length l. The set of all such ϕ is denoted as $\Phi_{i,p,t}$
$\delta(h)$, $\delta(\boldsymbol{h})$	$\delta(h)$ is an indicator function of h, which is equal to 1 when h is greater than 0, and 0 when h is smaller than or equal to 0. $\delta(\boldsymbol{h})$ is a vector, of which elements are $\delta(h)$
$\odot$, $\prod_\odot$	Element-wise product and cumulative element-wise product

ing ambiguity. Before the model training, elements of the weight matrix $\boldsymbol{W}^l$ are sampled from a probability distribution $P(w^l)$, such as a uniform or Gaussian distribution. The mean of the distribution is typically set to zero, and thus the variance, denoted by $var(w^l)$, determines the form of the distribution. Classic initialization methods, such as *Xavier* and *Lecun*, tend to stabilize variance across layers by setting an appropriate $var(w^l)$ for all layers. Stabilizing variance means equating $var(h^l)$ for $\forall l \in 0 \cdots L-1$, and equating $var(\frac{\partial Loss}{\partial h^l})$ for $\forall l \in 0 \cdots L-1$. In the context of CNNs, h^l denotes an element of a flattened feature map of the l_{th} layer. For example, to stabilize forward variance, *Lecun*, *Xavier* and *Kaiming* initialization set $var(w^l)$ to $\frac{1}{3m_1^{(l)}}$, $\frac{1}{m_1^{(l)}}$ and $\frac{2}{m_1^{(l)}}$ respectively. Meanwhile, *Xavier* and *Kaiming* initialization set $var(w^l)$ to $\frac{1}{m_2^{(l)}}$ and $\frac{2}{m_2^{(l)}}$ respectively to stabilize backward variance. Note that *Xavier* sets the final $var(w^l)$ to the harmonic average of $\frac{1}{m_1^{(l)}}$ and $\frac{1}{m_2^{(l)}}$, obtaining a trade-off between forward and backward variance stabilization. Classic initializations might lead to suboptimal performance for GNNs though they have been widely used by modern GNN architectures, since they disregard the impact of input graph structure and message mechanisms of GNN on variance. Furthermore, they implicitly assume that forward outputs and backward gradients of all neurons within each layer have the same variance. We next derive new variance expressions that directly take graph structure into account.

3. Forward and Backward Variance

In this section, we show an overview of derivations that finally provide analytic expressions of forward variance of GNNs defined by (1). The backward variance follows similar derivations. Then we present variance expressions in terms of $var(w^l)$ based on given derivations. Proofs of theorems and empirical verification of assumptions are presented in the Appendix A.

Firstly, to simplify subsequent derivations, we adopt a modification, proposed by (Xu et al., 2018b), to the activation function in the original GCN formulation (1). Given an arbitrary vector $\boldsymbol{x}$, we introduce an indicator function $\delta(\mathbf{x})$ that maps $\mathbf{x}$ into a binary vector, where each element is 1 if the corresponding element in $\mathbf{x}$ is greater than 0, and 0 otherwise. $\boldsymbol{z} = \sigma(\mathbf{x})$ is thereby rewritten by $\boldsymbol{z} = \delta(\boldsymbol{x}) \odot \boldsymbol{x}$. In this paper, we originally intend to investigate the influence of $\sigma(\mathbf{x})$ on $\boldsymbol{z}$. The nesting of the activation function and the input vector makes such investigation intractable. After the proposed modification decouples the nesting, we are able to investigate the influence of $\delta(\boldsymbol{x})$ and $\boldsymbol{x}$ on $\boldsymbol{z}$ separately, where $\delta(\boldsymbol{x})$ is a simple 0-1 vector. We therefore convert (1) into the following equations:

$$\begin{aligned} \hat{\boldsymbol{h}}_i^{l-1} &= \sum_{j \in \mathbb{N}(i)} d_{ij} \boldsymbol{h}_j^{l-1} \boldsymbol{W}^{l-1} \\ \boldsymbol{h}_i^l &= \delta(\hat{\boldsymbol{h}}_i^{l-1}) \odot \hat{\boldsymbol{h}}_i^{l-1}. \end{aligned} \tag{2}$$

3.1. The First Variance Decomposition

Based on (2), we now decompose the forward variance into the sum over variance of message propagation paths. As proposed by (Xu et al., 2018b; Gasteiger et al., 2022), we can decompose the acyclic GNN computation graph into a set of message propagation paths from the input to the l-th layer such that the embedding $\boldsymbol{h}_i^l$ can be viewed as a summation over message propagation paths of length l. The full details of how we conduct the decomposition are presented in Appendix C.

We denote an ordered sequence describing nodes along a message propagation path from node i to node j_{l-1} by $p = \{i, j_1, j_2, \cdots j_l\}$. The product set of neighbors $\{\mathbb{N}(i) \times \mathbb{N}(j_1) \cdots \mathbb{N}(j_{l-2}) \times \mathbb{N}(j_{l-1})\}$ is denoted by $\mathbb{N}(i, j_{l-1})$. The decomposition results in:

$$\boldsymbol{h}_i^l = \sum_{\substack{p \in \\ \mathbb{N}(i, j_{l-1})}} [(\prod_{\substack{k_1=0 \\ \odot}}^{l-1} \delta(\hat{\boldsymbol{h}}_{j_{k_1}}^{l-k_1-1}))(\prod_{k_2=0}^{l-1} d_{j_{k_2} j_{k_2+1}}) (\boldsymbol{h}_{j_l}^0 \prod_{k_3=0}^{l-1} \boldsymbol{W}^{k_3})]. \tag{3}$$

We denote a message propagation path of length l as $[\boldsymbol{h}_i^l]_p$, which is an element of the summation in (3). $[\boldsymbol{h}_i^l]_p$ is equal to:

$$[\boldsymbol{h}_i^l]_p = (\prod_{\substack{k_1=0 \\ \odot}}^{l-1} \delta(\hat{\boldsymbol{h}}_{j_{k_1}}^{l-k_1-1}))(\prod_{k_2=0}^{l-1} d_{j_{k_2} j_{k_2+1}})(\boldsymbol{h}_{j_l}^0 \prod_{k_3=0}^{l-1} \boldsymbol{W}^{k_3}). \tag{4}$$

$[\boldsymbol{h}_i^l]_p$ propagates input message from a l-hop neighbor j_l to the target node i along the path p. We denote the set including all such paths p as $\mathbb{P}_{i,l}$. Let $[h_i^l]_{p,t}$ be an element of the $[\boldsymbol{h}_i^l]_p$, then $var(h_i^l)$ is equal to $var(\sum_{p \in \mathbb{P}_{i,l}} [h_i^l]_{p,t})$, that is, the variance of node i is equal to the variance of the sum over all message propagation paths into node i. We know that $var(\sum_{p \in \mathbb{P}_{i,l}} [h_i^l]_{p,t})$ is equal to $\sum_{p \in \mathbb{P}_{i,l}} var([h_i^l]_{p,t}) + 2\sum_{p_1 \neq p_2} cov([h_i^l]_{p_1,t}, [h_i^l]_{p_2,t})$. We convert the covariance terms into the combination of variance terms $var([h_i^l]_{p,t})$ over different paths p, so that the variance of each node can be deduced from the variance of its message propagation paths. We observe that $[h_i^l]_{p_1,t}$ and $[h_i^l]_{p_2,t}$ have the multiplication of the same weight matrices, and δ terms only control whether or not a message propagation path is activated. Motivated by this observation, we make the following assumption to simplify the derivation and ease the conversion:

Assumption 3.1. Let p_1 and p_2 be two different elements of $\mathbb{P}_{i,l}$. Before model training, the Pearson correlation between $[h_i^l]_{p_1,t}$ and $[h_i^l]_{p_2,t}$ is approximately 1.

In Appendix A, we take the GNN following (2) and empirically investigate Pearson correlation of a set of message propagation paths on four datasets. All correlation results are greater than 0.83, which supports the feasibility of Assumption 3.1 for GNNs.

Based on the Assumption 3.1, the covariance term can be approximated by $\sqrt{var([h_i^l]_{p_1,t}) var([h_i^l]_{p_2,t})}$, $var(\sum_{p \in \mathbb{P}_{i,l}} [h_i^l]_{p,t})$ can thus be represented in terms of the combination of $var([h_i^l]_{p,t})$ over different paths p.

3.2. The Second Variance Decomposition

We now decompose the variance of each message propagation path into the sum over variances of its weight propagation paths. A weight propagation path of length l multiplies each element within weight matrices from the input layer to the l-th layer. Weight propagation paths are motivated by the forward propagation of MLP: There are many paths from neurons within the input layer to a certain neuron within the l-th layer, each of which propagates information through layers.

In (4), $\delta(\cdot)$ indicates if the path p is activated by ReLU, the multiplication of degree terms is a re-scaling constant, and $\boldsymbol{h}_{j_l}^0 \prod_{k_3=0}^{l-1} \boldsymbol{W}^{k_3}$ is a FNN with $\boldsymbol{h}_{j_l}^0$ as input. Moreover, (4) can be viewed as an FNN with activation function ReLU. As proposed by (Choromanska et al., 2015; Kawaguchi, 2016; Xu et al., 2018b; Gasteiger et al., 2022), we can decompose such FNN into a summation over weight propagation paths from the input layer to the l-th layer. To see this more formally, we first provide the expression of $[h_i^l]_{p,t}$ according to (4). Let $\mathbb{Z}(m)$ be a set $\{0, 1, \cdots, m\}$. We denote the set $\mathbb{Z}(m_1^{(0)}) \times \mathbb{Z}(m_1^{(1)}) \cdots \mathbb{Z}(m_1^{(l-1)})$ by $\hat{\mathbb{Z}}(l-1)$. Then we have $[h_i^l]_{p,t}$ equal to:

$$[h_i^l]_{p,t} = (\prod_{k_1=0}^{l-1} \delta(\hat{h}_{j_{k_1},t}^{l-k_1-1}))(\prod_{k_2=0}^{l-1} d_{j_{k_2} j_{k_2+1}}) (\sum_{\substack{(t_0 \cdots t_{l-1}) \in \\ \hat{\mathbb{Z}}(l-1)}} h_{j_l,t_0}^0 \prod_{k_3=0}^{l-1} w_{t_{k_3},t_{k_3+1}}^k). \tag{5}$$

$\delta(\hat{h}_{j_{k_1},t}^{l-k_1-1})$ in (5) is the t_{th} element of $\delta(\hat{\boldsymbol{h}}_{j_{k_1}}^{l-k_1-1})$ in (4), and the $\sum_{t_0 \cdots t_{l-1}} h_{j_l,t_0}^0 \prod_{k_3} w_{t_{k_3},t_{k_3+1}}^k$ in (5) is the t_{th} output element of the FNN $\boldsymbol{h}_{j_l}^0 \prod_{k_3=0}^{l-1} \boldsymbol{W}^{k_3}$ in (4).To simplify (5), let δ_p be $\prod_{k_1} \delta(\hat{h}_{j_{k_1},t}^{l-k_1-1})$, and d_p be $\prod_{k_2} d_{j_{k_2} j_{k_2+1}}$. An output element of the FNN can be viewed as summation over weight propagation paths, where each weight propagation path, denoted as $[h_i^l]_{p,t,\phi}$, is:

$$[h_i^l]_{p,t,\phi} = h_{j_l,t_0}^0 w_{t_0,t_1}^0 \cdots w_{t_{l-1},t}^{l-1}, \tag{6}$$

where ϕ equals to $\{t_0 \cdots t_{l-1}, t\}$ is an ordered sequence describing neurons along that weight propagation path. We denote the set including all such ϕ as $\Phi_{i,p,t}$. $[h_i^l]_{p,t,\phi}$ multiplies elements of input features and weight matrices across l layers. It is obvious that $var([h_i^l]_{p,t})$ is equal to $d_p^2 var(\delta_p \sum_{\phi \in \Phi_{i,p,t}} [h_i^l]_{p,t,\phi})$. For this formula, we intend to extract δ_p from it then convert the variance of sum into the sum over variance of $[h_i^l]_{p,t,\phi}$, so that variance of each message propagation path can be expressed by the variance of weight propagation paths. To achieve this conversion, we hold the following three assumptions:

Assumption 3.2. Prior to model training, let ϕ_1 and ϕ_2 be two different elements of $\Phi_{i,p,t}$. Following assumptions proposed by (He et al., 2015), we assume the Pearson correlation between $\delta_p[h_i^l]_{p,t,\phi_1}$ and $\delta_p[h_i^l]_{p,t,\phi_2}$ is approximately 0.

Assumption 3.3. Prior to model training, the Pearson correlation between δ_p and $[h_i^l]_{p,t,\phi}$ is approximately 0, and the Pearson correlation between δ_p^2 and $[h_i^l]_{p,t,\phi}^2$ is also approximately 0.

Assumption 3.4. (Xu et al., 2018b; Choromanska et al., 2015; Kawaguchi, 2016) assume δ_p to be a Bernoulli random variable, and different δ_p with the same length of p have the same success probability. Inspired by these given assumptions, we assume such success probability to be 0.5^l before the model training, where l is the length of p.

When Assumptions 3.2 to 3.4 hold, $var(\delta_p \sum_{\phi \in \Phi_{i,p,t}})$ can be converted into $0.5^l var([h_i^l]_{p,t,\phi})$. According to (6), we are able to express $var(h_i^l)$ in terms of $var(w^k)$. Analogously, $var(\frac{\partial Loss}{\partial h_i^l})$ can also be represented in terms of $var(w^k)$. Specifically, the expressions of variance are provided as follows:

Theorem 3.5. *Given an L-layer GNN defined by (2), we define $\widetilde{\boldsymbol{A}}$ as a matrix in the sense that $\widetilde{\boldsymbol{A}}_{ij}$ is equal to d_{ij} if node i and j are connected, and 0 otherwise. Let $\boldsymbol{h}^0$ be $[\mathcal{M}(\boldsymbol{h}_0^0) \cdots \mathcal{M}(\boldsymbol{h}_{|\mathbb{N}|-1}^0)]^T$, where $\mathcal{M}(\boldsymbol{v})$ denotes the mean over elements of the vector $\boldsymbol{v}$. Also, $[\boldsymbol{v}]_i^2$ denotes the square of the i_{th} element of the vector $\boldsymbol{v}$. Then if Assumptions 3.1 to 3.4 hold, $var(h_i^l)$ is equal to:*

$$var(h_i^l) = (\frac{\prod_{k_1=0}^{l-1} m_1^{(k_1)}}{2^l})(\prod_{k_2=0}^{l-1} var(w^{k_2}))([\widetilde{\boldsymbol{A}}^l \boldsymbol{h}^0]_i^2). \quad (7)$$

Next, to derive the formula of $var(\frac{\partial Loss}{\partial h_i^l})$, we need one additional lemma and assumption.

Lemma 3.6. *Let $y(i)$ denote the label of node i and $s(\boldsymbol{v})_i$ denote the i_{th} element of the softmax of the vector $\boldsymbol{v}$. Then given a cross-entropy loss as in Section 2.1, we have:*

$$\frac{\partial Loss}{\partial h_{i,t}^L} = \begin{cases} \frac{s(\boldsymbol{h}_i^L)_t - 1}{|\mathbb{N}|} & t = y(i) \\ \frac{s(\boldsymbol{h}_i^L)_t}{|\mathbb{N}|} & t \neq y(i). \end{cases} \quad (8)$$

Assumption 3.7. Prior to model training, we assume the random uniformity of predicted labels at initialization and thereby $s(\boldsymbol{h}_i^L)_t$ in Lemma 3.6 is equal to $1/C$, where C is the output dimension of the last GNN layer.

The expression of $var(\frac{\partial Loss}{\partial h_i^l})$ is presented as the following theorem.

Theorem 3.8. *Let $\mathbf{1} \in \mathbb{R}^{|\mathbb{N}|}$ be the vector with all 1s. Assuming the same conditions as in Theorem 3.5, and the additional Assumption 3.7, $var(\frac{\partial Loss}{\partial h_i^l})$ is equal to:*

$$\begin{aligned} var(\frac{\partial Loss}{\partial h_i^l}) =& (\frac{\prod_{k_1=l+1}^{L-1} m_2^{(k_1)}(C-1)}{2^{L-l}|\mathbb{N}|^2 C}) \\ & (\prod_{k_2=l+1}^{L-1} var(w^{k_2}))([\widetilde{\boldsymbol{A}}^{L-l}\mathbf{1}]_{\mathbf{i}}^{\mathbf{2}}). \end{aligned} \quad (9)$$

From Theorems 3.5 and 3.8, we observe that nodes at each layer have different variance since nodes have different receptive fields expressed by $[\widetilde{\boldsymbol{A}}^{L-l}\mathbf{1}]_{\mathbf{i}}^{\mathbf{2}}$. This finding breaks the assumption of classic methods that all neurons at each layer have the same variance. Furthermore, while classic methods only consider the impact of hidden dimension and activation function on variance, we can see that variance is also affected by the graph structure, the message propagation of GNNs, input features and the number of nodes. Specifically, in formulas of both theorems, the constant 2 is computed based on the *ReLU* activation function following (He et al., 2015), the constants $m_1^{(k_1)}$, $m_2^{(k_1)}$, C are input or output dimensions of weight matrices, $|\mathbb{N}|$ is the number of nodes, $\widetilde{\boldsymbol{A}}$ is the renormalized adjacent matrix, which is determined by the graph structure as well as the message propagation mechanism of the GNN, and $\boldsymbol{h}^0$ is determined by input features of the given graph. We now apply these insights towards the development of a new initialization method.

4. Proposed Virgo Initialization

To stabilize variance for GNNs, we propose a initialization method named Virgo which incorporates the factors mentioned in the last section. The target of Virgo is to make $\sum_i var(h_i^l)$ equal to $\sum_i var(h_i^{l+1})$, and $\sum_i var(\frac{\partial Loss}{\partial h_i^l})$ equal to $\sum_i var(\frac{\partial Loss}{\partial h_i^{l+1}})$. Consideration of these two conditions then leads to the following two theorems:

Theorem 4.1. *Assuming the same conditions as in Theorem 3.5, to make $\sum_i var(h_i^l)$ equal to $\sum_i var(h_i^{l+1})$, we require that:*

$$var(w^l) = \frac{2}{m_1^{(l)}} \frac{\mathbf{1^T}[\widetilde{\mathbf{A}}^{\mathbf{l-1}}\mathbf{h^0}]^{\mathbf{2}}}{\mathbf{1^T}[\widetilde{\mathbf{A}}^{\mathbf{l}}\mathbf{h^0}]^{\mathbf{2}}}. \quad (10)$$

Theorem 4.2. *Assuming the same conditions as in Theorem 3.8, to make $\sum_i var(\frac{\partial Loss}{\partial h_i^l})$ equal to $\sum_i var(\frac{\partial Loss}{\partial h_i^{l+1}})$, we require that:*

$$var(w^l) = \frac{2}{m_2^{(l)}} \frac{\mathbf{1^T}[\widetilde{\mathbf{A}}^{\mathbf{L-l-1}}\mathbf{1}]^{\mathbf{2}}}{\mathbf{1^T}[\widetilde{\mathbf{A}}^{\mathbf{L-l}}\mathbf{1}]^{\mathbf{2}}}. \quad (11)$$

$var(w^l)$ as calculated by Theorems 4.1 and 4.2 stabilizes forward and backward variances respectively. Within the variance expressions of these two theorems, the constants 2, $m_1^{(k_1)}$, and $m_2^{(k_1)}$ are introduced to mitigate the impact of the activation function and hidden dimension on variance, analogous to factors considered by classic methods. In constrast, the appearance of $\widetilde{\boldsymbol{A}}$ and $\boldsymbol{h}^0$ encapsulate the innovative part of Virgo, which is taking graph structure, message passing and input features into account to better stabilize the variance of GNNs.

5. Experiments

In this section, we conduct experiments to investigate the performance of Virgo. Firstly, we evaluate several models on three popular graph tasks, node classification (Section 5.1), link prediction (Section 5.2) and graph classification (Section 5.3), to showcase the performance and generalizability of Virgo. Then to provide further insights, we test the variance stability (Section 5.4) of models trained with Virgo. In experiments below, we conduct hyperparameter sweep to search for best hyperparameter settings. To be specific, for each hyperparameter setting, we calculate the mean and standard deviation of 10 trials across different random seeds. We iterate over multiple hyperparameter settings and search for the setting with the best mean on validation datasets. We then report the mean and standard deviation on testing datasets with the selected setting as the final results. All experiments are conducted on a single Tesla T4 GPU with 16GB memory. Details of experimental setting are presented in Appendix B

Baseline Initializations We compare Virgo with (i) *Lecun* initialization, designed for stabilizing the forward variance of linear FNNs; (ii) *Xavier* initialization, for stabilizing both forward and backward variances of linear FNNs; and (iii) *Kaiming* initialization, which proposes two methods that stabilize either the forward or backward variance of CNNs activated by ReLU. We denote the two *Kaiming* variants as KaiFor and KaiBack, respectively. Similarly, we denote model initialization following Theorem 4.1 as VirgoFor, and Theorem 4.2 as VirgoBack, which stabilizes forward and backward variance respectively.

GNN Architectures We evaluate the model performance on node classification, link prediction and graph classification tasks with some classic GNN architectures: (i) GCN (Kipf & Welling, 2016), a spectral-based GNN for semi-supervised node classification tasks; (ii) GraphSAGE (Hamilton et al., 2017), stacking spatial-based convolutions to propagate message over graphs; (iii) GIN (Xu et al., 2018a), mitigating the incapability of GNNs to distinguish different graphs structures; (iv) NGNN (Song et al., 2021) variants of (i) and (ii), which deepens GNN models with additional MLP layers interspersed with graph propagation. Note that with NGNN variants, we are able to investigate the capability of Virgo initializations on deeper GNNs more fairly while largely avoiding the effects of over-smoothing and over-squashing on model performance, i.e., Virgo is not presently designed to alleviate oversmoothing or oversquashing on deep GNNs. In practice, Virgo is directly applied to GNN layers of NGNN. All models are implemented with DGL (Wang et al., 2020) and PyG (Fey & Lenssen, 2019).

Datasets For node classification, we choose three citation network datasets (Sen et al., 2008): cora, citeseer, pubmed, and three OGB (Hu et al., 2020) datasets: ogbn-arxiv, ogbn-proteins and ogbn-products. For link prediction, we adopt four OGB datasets: ogbl-ddi, ogbl-collab, ogbl-citation2 and ogbl-ppa. For graph classification, we take three social network datasests imdb_b, imdb_m and collab from (Yanardag & Vishwanathan, 2015), and two OGB datasets ogbg-molhiv and ogbg-molpcba.

5.1. Node Classification

Experimental setting We use DGL to implement GCN on cora, citeseer and pubmed, and take implementations of the OGB team on the OGB node classification leaderboard to implement GCN on ogbn-arxiv and ogbn-proteins, and GraphSAGE on ogbn-products. We use neighbor sampling to support mini-batch training of GraphSAGE on ogbn-products. We tune hyper-parameters as specified previously and compare model performance of GCN and GraphSAGE with different initialization methods, including *Xavier*, *Lecun*, KaiFor, KaiBack, VirgoFor, and VirgoBack. We use the Adam (Kingma & Ba, 2014) optimizer to update trainable parameters and use early-stop mechanism to reduce the training time overhead.

Results The evaluation results are presented in Table 2. We observe that models with Virgo outperform models with other initializations on 5 out of 6 datasets, the lone exception being pubmed. And even on pubmed, VirgoFor and VirgoBack obtain the second and third best performance among all initializations. Furthermore, models with Virgo perform well on larger size graphs (ogbn-proteins and ogbn-products). For example, GCN initialized by Virgo on ogbn-proteins produces the largest performance gain; specifically, VirgoBack has 1.04% higher accuracy than the best baseline initialization method *Xavier*.

5.2. Link Prediction

Experimental setting We adopt the implementations of the OGB team on the OGB link prediction leaderboard for GCN and GraphSAGE, and use DGL to implement their NGNN variants. We take *Xavier* and *Lecun* as baseline initializations. Baselines on the leaderboard take *Xavier* or *Lecun* to initialize models. We observe that the reported numbers of many leaderboard submissions of baselines are significantly lower than model performance with Virgo, which we believe is in part an artifact of the fact that baselines on the OGB

Table 2. The performance of GCN on cora, citeseer, pubmed, ogbn-arxiv and ogbn-proteins, GraphSAGE with neighbor sampling on ogbn-products. The numbers indicating the first and second place of mean accuracy are highlighted in red and blue respectively.

Methods	Datasets					
	cora	citeseer	pubmed	arxiv	proteins	products$_{sage}$
KaiFor	81.33±0.46	70.14±0.47	78.92±0.40	71.78±0.44	73.51±0.30	78.80±0.28
KaiBack	81.57±0.43	70.79±0.49	79.20±0.55	71.44±0.37	73.41±0.58	78.73±0.24
Lecun	81.41±0.33	70.97±0.43	79.48±0.31	71.82±0.24	73.29±0.44	78.49±0.37
Xavier	81.50±0.20	71.09±0.52	79.10±0.37	71.74±0.32	73.54±0.67	78.89±0.31
VirgoFor	82.14±0.52	71.96±0.47	79.42±0.42	72.22±0.17	74.41±0.43	79.50±0.36
VirgoBack	82.14±0.48	71.36±0.50	79.34±0.22	72.18±0.34	74.58±0.53	79.45±0.36

leaderboard may not be sufficiently trained. For example, GCN on ogbl-ddi achieves 37.07 hit@20 reported on leaderboard, surprisingly worse than GCN with Virgo (67.98 and 74.83). We observe that the number of training epochs of leaderboard submissions is too small to allow model training to converge. Their validation accuracy curves are still rising rather than staying flat until the end of the model training. Therefore, we re-train baselines on link prediction datasets with more epochs (for example, we take around 1000 to 2000 epochs for models on ogbl-ddi compared to 80 to 400 epochs taken by leaderboard submissions) and carefully tune their hyperparameters for a fair comparison with Virgo. The hyperparameter tuning setting is the same for baselines and Virgo.

The evaluation metrics for models on ogbl-ddi, ogbl-collab, ogbl-citation2 and ogbl-ppa are hits@20, hits@50, mrr and hits@100, respectively. We use Adam optimizer to train models and pick the model checkpoint which has the best performance on the validation dataset. The selected checkpoint is then used to evaluate on the test dataset.

Results The results are presented in Table 3. We observe that Virgo leads to better model performance relative to other initializations in most cases. For example, NGNN-GraphSAGE with Virgo (the best one is VirgoFor with performance 80.36%) exhibits the largest performance improvement (7.31%) relative to NGNN-GraphSAGE with baseline initializations (the best one is *Xavier* with performance 73.07%). Overall the model performance with Virgo is best in 14 out of 16 cases. In other cases where Virgois not the top 1 (NGNN-GCN and NGNN-GraphSAGE on ogbl-ppa), Virgostill achieves second place. Furthermore, we see that Virgo improves performance of NGNN variants, which have around 3 GCN/GraphSAGE layers and 2 MLP layers, in most cases, indicating that Virgo can be helpful to deep GNNs.

5.3. Graph Classification

Experimental setting We use DGL to implement models on imdb_b, imdb_m and collab, and we take implementations of the OGB team on the OGB graph classification leaderboard to conduct experiments on OGB datasets. We evaluate GCN and GIN with *Xavier*, *Lecun* and Virgo. To compute the initial variance of weight matrices based on Virgo, we sample a subset of training graphs into a single graph defined as the approximation graph, and utilize its graph structure to approximate $\widetilde{\boldsymbol{A}}$ in Theorems 4.1 and 4.2. For imdb_b, imdb_m, collab and ogbg-molhiv, we merge all training graphs into the approximation graph. For ogbg-molpcba, we randomly and uniformly sample 10,000 training graphs and merge them into the approximation graph since its training dataset is too large to be fed into a single GPU. We use Adam optimizer to update trainable parameters. For OGB datasets, we take the model checkpoint that has the best validation performance, and evaluate this checkpoint on test datasets to obtain the test performance. We evaluate classification accuracy on imdb and collab, roc-auc on ogbg-molhiv, and average precision on ogbg-molpcba.

Results The experimental results are reported in Table 4. We have three observations: First, both VirgoFor and VirgoBack take top 2 in 8 out of 10 cases, and 9 out of 10 cases have Virgo as their best initialization method. These results shows the benefit of Virgo to model performance, which is not limited to node classification and link prediction tasks. In other cases where Virgo does not occupy top 2 positions (GCN on imdb_b and ogbg-molhiv), it takes at least the second place. Second, by comparing the best model performance with Virgo and the best one with baseline initializations, Virgo brings at least 1% improvements to GCN on 4 out of 5 datasets: imdb_b (1.23%), collab (1.53%), ogbg-molhiv (1.72%) and ogbg-molpcba (1.11%). Finally, performance improvements on ogbg-molpcba with trivial sampling methods indicate that we can simply use random uniform sampling methods as described previously to achieve competitive performance with Virgo.

Table 3. The performance of GCN, GraphSAGE and their NGNN variants on link prediction tasks. The numbers indicating the first and second place of the average of evaluation metrics are highlighted in red and blue respectively.

Models	Methods	Datasets ddi	collab	citation2	ppa
GCN	Lecun	69.77±10.62	52.23±0.34	68.04±2.29	39.35±1.29
	Xavier	55.16±10.64	53.64±0.25	80.88±0.18	37.40±0.66
	VirgoFor	67.98±11.91	54.58±0.51	81.05±0.16	39.38±1.03
	VirgoBack	74.83±10.49	54.31±0.42	81.12±0.23	39.85±0.89
NGNN-GCN	Lecun	58.18±10.21	51.93±0.63	54.38±2.71	44.26±3.48
	Xavier	67.71±11.90	52.97±0.50	81.07±0.12	45.47±1.64
	VirgoFor	69.05±9.54	54.16±0.51	81.41±0.28	45.10±1.54
	VirgoBack	65.32±8.78	54.13±0.38	81.36±0.23	46.99±0.54
GraphSAGE	Lecun	71.26±13.79	53.62±0.44	82.62±0.12	42.98±2.38
	Xavier	70.86±10.94	53.48±0.34	83.39±0.11	41.99±2.42
	VirgoFor	72.73±7.58	53.67±0.74	83.74±0.01	42.45±0.66
	VirgoBack	72.48±9.50	54.16±0.47	83.49±0.14	43.02±0.56
NGNN-GraphSAGE	Lecun	65.77±13.19	52.14±0.43	81.61±0.07	42.41±1.48
	Xavier	73.07±8.57	53.59±0.38	83.44±0.07	44.36±1.26
	VirgoFor	80.36±4.35	54.37±0.24	83.13±0.13	44.09±0.06
	VirgoBack	76.02±10.21	53.87±0.19	83.36±0.12	43.95±1.69

5.4. Variance Stability

In this section, we compare variance stability of GCN following (2) on ogbn-arxiv and ogbn-proteins at initialization with different methods. For each combination of an initialization method and a dataset, we pick the best hyperparameter setting that has been investigated in Section 5.1, and test forward and backward variance across 5 layers. Specifically, we compute the variance of each node, and compute the mean of variance over nodes at each layer. The results are presented in Figure 1. We observe that Virgo leads to more stable variance change than classic methods. For example, in the cases of backward variance on ogbn-arxiv and forward variance on ogbn-proteins, only Virgo mitigates the steep decline in variances towards zero, emblematic of the importance of accounting for graph structure and message passing relative to other factors in stabilizing the variance.

6. Related Work

Our work is closely related to Graph Neural Networks (GNNs) and initialization methods. We introduce classic work in these fields as follows.

Graph Neural Networks There are a number of approaches that generalize convolution operations on images to the graph domain and achieve state-of-the-art performance on popular tasks of graphs, such as node classification, link prediction and graph classification. (Defferrard et al., 2016; Levie et al., 2018; Kipf & Welling, 2016) propose spectral-based convolutional neural networks based on graph Laplacian matrix for learning on graphs. (Wu et al., 2019) achieves comparable peformance with GCN (Kipf & Welling, 2016) while reduce excess complexity. (Hamilton et al., 2017) propose spatial-based convolutions to propagate message over graphs based on nodes' spatial dependencies. (Veličković et al., 2017; Brody et al., 2021) utilize attention mechanism to learn contributions of neighboring nodes to the target nodes. (Xu et al., 2018a) investigates the expressive power of classic GNNs and develop a simple but more powerful structure. (Xu et al., 2018b; Li et al., 2019) are designed to learn higher level of knowledge from graphs by increasing the number of GCN layers. Inspired by (Lin et al., 2013; Xu et al., 2018a), (Song et al., 2021) equips each GCN layer with multiple linear layers to form a NGNN block, which extracts more complex semantics from graphs.

Initialization methods Several initialization methods have been proposed to define initial values of model parameters prior to the model training. *Lecun* initialization (LeCun et al., 2012) requires forward variance to be 1. *Xavier* initialization (Glorot & Bengio, 2010) is similar to *Lecun* but considers both forward and backward variance. Despite *Lecun* and *Xavier* assuming no non-linearity in neural networks, they work well in many applications. *Kaiming* initialization (He et al., 2015) extends *Xavier* to CNNs with ReLU non-linearity. (Saxe et al., 2013) exhibit a new class of orthogonal matrix initialization for deep linear neural

Table 4. The performance of GCN and GIN on graph classification tasks. The numbers indicating the first and second place of the average of evaluation metrics are highlighted in red and blue respectively.

Models	Methods	Datasets imdb$_b$	imdb$_m$	collab	molhiv	molpcba
GCN	Lecun	74.57±2.19	52.04±1.77	82.43±0.84	77.06±0.63	20.44±0.21
	Xavier	74.17±2.12	51.83±3.23	82.17±1.13	76.49±0.95	20.62±0.31
	VirgoFor	75.80±2.32	52.13±2.00	83.96±0.78	77.94±0.58	21.12±0.50
	VirgoBack	75.40±4.84	52.67±2.15	83.44±0.73	78.78±0.16	21.73±0.15
GIN	Lecun	75.07±2.43	52.61±1.41	83.10±0.74	77.12±1.01	23.45±0.23
	Xavier	75.42±3.31	52.24±1.57	83.27±1.39	76.16±1.76	23.01±0.33
	VirgoFor	75.20±3.66	53.20±1.36	84.32±0.63	77.09±1.19	24.15±0.49
	VirgoBack	74.60±2.73	53.60±2.00	83.84±1.17	77.90±1.43	24.23±0.12

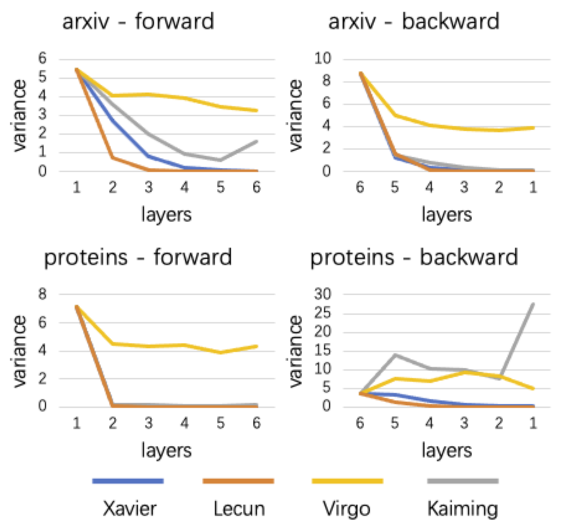

Figure 1. Variance stability of GCN on ogbn-arxiv and ogbn-proteins at the initialization. *Upper left*: Forward variance on ogbn-arxiv($\times 10^{-2}$). *Upper right*: Backward variance on ogbn-arxiv($\times 10^{-25}$). *Bottom left*: Forward variance on ogbn-proteins ($\times 10^{-3}$). *Bottom right*: Backward variance on ogbn-proteins($\times 10^{-16}$). As for Kaiming and Virgo, we adopt KaiFor and VirgoFor in figures labeled *forward*, and KaiBack and VirgoBack in figures labeled *backward*.

networks. (Mishkin & Matas, 2015) proposes *LSUV* initialization based on (Saxe et al., 2013) to consider the impact of more model components on variance, such as tanh and maxout. (Sussillo & Abbott, 2014) designs a *Random Walk* initialization for FNNs with non-linearity to keep constant the logarithm of squared magnitude of gradients acorss all layers. (Jaiswal et al., 2022) proposes a topology-aware isometric initialization to facilitate gradient flow during GCN training. (Han et al., 2022) initialize GNNs with pre-trained MLP parameters to train GNNs more efficiently. However, most existing work does not directly apply to GNNs because of: only analyzing output and gradients with FNNs (LeCun et al., 2012; Glorot & Bengio, 2010; He et al., 2015; Saxe et al., 2013; Mishkin & Matas, 2015; Sussillo & Abbott, 2014), ignoring the impact of non-linearities (LeCun et al., 2012; Glorot & Bengio, 2010; Saxe et al., 2013; Jaiswal et al., 2022), assuming outputs and gradients of neurons at each layer are i.i.d (LeCun et al., 2012; Glorot & Bengio, 2010; He et al., 2015). In contrast, our analysis is explicitly based on GNNs over graphs, and accounts for the fact that the outputs and gradients of different nodes have correlated variances due to the effective receptive fields at each layer. We then exploit these findings to develop Virgo, which is better equipped to stabilize GNN variances.

7. Conclusion

In this paper, we derive explicit expressions for the forward and backward variance of GNN initializations, and analyze deficiencies of classic initialization methods when applied to stabilizing them. Informed by this perspective, we propose a new GNN initialization scheme Virgo, and conduct comprehensive experiments to compare with 4 classic initialization methods on 15 datasets across 3 popular graph learning tasks showing superior performance.

In the future, there are two shortcomings of Virgothat could potentially be addressed: (i) Virgo is derived based on GNNs that have pre-computed constant coefficients between neighboring nodes, and thus it cannot directly be generalized to GNNs like GAT (Veličković et al., 2017) with adaptive/learnable coefficients between neighbors. (ii) Virgo only considers one level of aggregation, thus it is not yet suitable for models like RGCN (Schlichtkrull et al., 2018) that have multiple levels of aggregation. Beyond these considerations, we do not believe that our approach will have any undue negative societal impact beyond the minor potential essentially shared by all GNN methods, e.g., propagating unfair biases, etc.

References

Brody, S., Alon, U., and Yahav, E. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*, 2021.

Choromanska, A., Henaff, M., Mathieu, M., Arous, G. B., and LeCun, Y. The loss surfaces of multilayer networks. In *Artificial intelligence and statistics*, pp. 192–204. PMLR, 2015.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29, 2016.

Fan, W., Ma, Y., Li, Q., He, Y., Zhao, E., Tang, J., and Yin, D. Graph neural networks for social recommendation. In *The World Wide Web Conference*, pp. 417–426, 2019.

Fey, M. and Lenssen, J. E. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

Gasteiger, J., Qian, C., and Günnemann, S. Influence-based mini-batching for graph neural networks. *arXiv preprint arXiv:2212.09083*, 2022.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010.

Hamilton, W. L., Ying, R., and Leskovec, J. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 1025–1035, 2017.

Han, X., Zhao, T., Liu, Y., Hu, X., and Shah, N. Mlpinit: Embarrassingly simple gnn training acceleration with mlp initialization. *arXiv preprint arXiv:2210.00102*, 2022.

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

Jaiswal, A., Wang, P., Chen, T., Rousseau, J., Ding, Y., and Wang, Z. Old can be gold: Better gradient flow can make vanilla-gcns great again. *Advances in Neural Information Processing Systems*, 35:7561–7574, 2022.

Jing, X. and Xu, J. Fast and effective protein model refinement using deep graph neural networks. *Nature Computational Science*, 1(7):462–469, 2021.

Kawaguchi, K. Deep learning without poor local minima. *Advances in neural information processing systems*, 29, 2016.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K.-R. Efficient backprop. In *Neural networks: Tricks of the trade*, pp. 9–48. Springer, 2012.

Levie, R., Monti, F., Bresson, X., and Bronstein, M. M. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Transactions on Signal Processing*, 67(1):97–109, 2018.

Li, G., Muller, M., Thabet, A., and Ghanem, B. Deepgcns: Can gcns go as deep as cnns? In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 9267–9276, 2019.

Lin, M., Chen, Q., and Yan, S. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

Liu, Z., Dou, Y., Yu, P. S., Deng, Y., and Peng, H. Alleviating the inconsistency problem of applying graph neural network to fraud detection. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 1569–1572, 2020.

Mishkin, D. and Matas, J. All you need is a good init. *arXiv preprint arXiv:1511.06422*, 2015.

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.

Nguyen, T., Le, H., Quinn, T. P., Le, T., and Venkatesh, S. Predicting drug–target binding affinity with graph neural networks. *BioRxiv*, pp. 684662, 2020.

Rossi, E., Chamberlain, B., Frasca, F., Eynard, D., Monti, F., and Bronstein, M. Temporal graph networks for deep learning on dynamic graphs. *arXiv preprint arXiv:2006.10637*, 2020.

Saxe, A. M., McClelland, J. L., and Ganguli, S. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

Schlichtkrull, M., Kipf, T. N., Bloem, P., Berg, R. v. d., Titov, I., and Welling, M. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pp. 593–607. Springer, 2018.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

Song, X., Ma, R., Li, J., Zhang, M., and Wipf, D. P. Network in graph neural network. *arXiv preprint arXiv:2111.11638*, 2021.

Strokach, A., Becerra, D., Corbi-Verge, C., Perez-Riba, A., and Kim, P. M. Fast and flexible protein design using deep graph neural networks. *Cell systems*, 11(4):402–411, 2020.

Sussillo, D. and Abbott, L. Random walk initialization for training very deep feedforward networks. *arXiv preprint arXiv:1412.6558*, 2014.

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

Wang, J., Wen, R., Wu, C., Huang, Y., and Xion, J. Fdgars: Fraudster detection via graph convolutional networks in online app review system. In *Companion Proceedings of The 2019 World Wide Web Conference*, pp. 310–316, 2019.

Wang, M., Zheng, D., Ye, Z., Gan, Q., Li, M., Song, X., Zhou, J., Ma, C., Yu, L., Gai, Y., Xiao, T., He, T., Karypis, G., Li, J., and Zhang, Z. Deep graph library: A graph-centric, highly-performant package for graph neural networks, 2020.

Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. Simplifying graph convolutional networks. In *International conference on machine learning*, pp. 6861–6871. PMLR, 2019.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018a.

Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K., and Jegelka, S. Representation learning on graphs with jumping knowledge networks. *CoRR*, abs/1806.03536, 2018b.

Yanardag, P. and Vishwanathan, S. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1365–1374, 2015.

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.

Yu, W., Lin, X., Liu, J., Ge, J., Ou, W., and Qin, Z. Self-propagation graph neural network for recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

A. Proof and Empirical Verification

In this section, we show empirical verification of assumptions, and proofs of one lemma and multiple theorems in the main body of this paper. All experiments are conducted on a 3-layer GCN initialized by *Xavier* initialization. We argue that the initialization method does not affect empirical results in this section since they are not related to distributions of GCN weight matrices.

Assumption 3.1 We first take a forward propagation of GCN on the input graph and obtain hidden embeddings of 3 layers. Then we randomly sample two nodes u and v from the input graph, and take all paths of length 3 from u to v in the graph if there exists at least one. The random sampling process continues until the number of paths reach 100. Let's recap that the expression of message propagation paths is:

$$[\boldsymbol{h}_i^l]_p = (\prod_{\substack{k_1=0 \\ \odot}}^{l-1} \delta(\hat{\boldsymbol{h}}_{j_{k_1}}^{l-k_1-1}))(\prod_{k_2=0}^{l-1} d_{j_{k_2}j_{k_2+1}})(\boldsymbol{h}_{j_l}^0 \prod_{k_3=0}^{l-1} \boldsymbol{W}^{k_3}) \tag{12}$$

It's obvious that sampled paths are instances of p that has length 3 in (12). For (12), the node v and u of each sampled path are the destination node i and the starting node j_l of the p, respectively. We apply δ to pre-activated embeddings of nodes to estimate $\delta(\hat{\boldsymbol{h}}_{j_{k_1}}^{l-k_1-1})$, then calculate the multiplication of resulted quantities along each path to estimate $\prod_{k_1,\odot} \delta(\hat{\boldsymbol{h}}_{j_{k_1}}^{l-k_1-1})$. We take the multiplication of degrees of nodes along the path p as $\prod_{k_2=0}^{l-1} d_{j_{k_2}j_{k_2+1}}$, and take the multiplication of weight matrices across 3 GCN layers with input features of node j_l as $\boldsymbol{h}_{j_l}^0 \prod_{k_3=0}^{l-1} \boldsymbol{W}^{k_3}$. As a result, we are able to estimate $[\boldsymbol{h}_i^l]_p$ with quantities mentioned above. Next, we randomly sample 100 neurons from each $[\boldsymbol{h}_i^l]_p$ as its 100 $[\boldsymbol{h}_i^l]_{p,t}$, and calculate the Pearson correlation between each pair of $[\boldsymbol{h}_i^l]_{p_1,t}$ and $[\boldsymbol{h}_i^l]_{p_2,t}$, where p_1 is not equal to p_2. Thereby we obtain 4950(pairs) resulted numbers, which indicate the Pearson correlation between message propagation paths of length 3. We take the average and standard deviation of them, and put results in Table 5.

In Table 5, we show evaluation results of 3-layer GCN on cora, pubmed, citeseer and ogbn-arxiv. We test the Pearson correlation of message propagation paths that have length 1, 2 besides 3. The row titled with **Expected** indicates that we require the Pearson correlation to be 1 to hold Assumption 3.1. We can see that the Pearson correlation on all datasets are greater than 0.83, especially on ogbn-arxiv where results are greater than 0.9.

Table 5. Estimation of the Pearson correlation between different message propagation paths.

Dataset	Layers 1	2	3
cora	0.83±0.13	0.86±0.12	0.84±0.15
pubmed	0.89±0.09	0.89±0.09	0.93±0.06
citeseer	0.83±0.12	0.85±0.11	0.85±0.12
arxiv	0.91±0.06	0.91±0.05	0.94±0.04
Expected	1.0	1.0	1.0

Assumption 3.2 We use the same experiment setting as in the empirical verification of Assumption 3.1. For each $\delta_p[\boldsymbol{h}_i^l]_{p,t}$, we randomly take 50 neurons as observations of $\delta_p[h_i^l]_{p,t,\phi_1}$ and take remain 50 neurons as observations of $\delta_p[h_i^l]_{p,t,\phi_2}$. We have 4950(pairs) of $\delta_p[h_i^l]_{p,t,\phi_1}$ and $\delta_p[h_i^l]_{p,t,\phi_2}$, then we calculate the Pearson correlation between each pair. We take the average and standard deviation of resulted numbers and report results in Table 6.

In Table 6, the row titled with **Expected** indicates that we require the Pearson correlation to be 0 to hold Assumption 3.2. We observe that all average numbers of the Pearson correlation are less than 0.1.

Assumption 3.3 We use the same experiment setting as in the empirical verification of Assumption 3.1. Let's recap

Table 6. Estimation of the Pearson correlation between different weight propagation paths.

Dataset	Layers 1	2	3
cora	0.07±0.05	0.07±0.07	0.05±0.07
pubmed	0.06±0.04	0.07±0.06	0.04±0.06
citeseer	0.07±0.05	0.06±0.05	0.03±0.04
arxiv	0.07±0.06	0.08±0.07	0.07±0.06
Expected	0.0	0.0	0.0

expressions of δ_p and $[h_i^l]_{p,t,\phi}$:

$$\delta_p = \prod_{k=0}^{l-1} \delta(\hat{h}_{j_k,t}^{l-k-1}) \tag{13}$$

$$[h_i^l]_{p,t,\phi} = h_{j_l,t_0}^0 w_{t_0,t_1}^0 \cdots w_{t_{l-1},t}^{l-1} \tag{14}$$

For each message propagation path p, we take the estimation of neurons of $\prod_{k_1,\odot} \delta(\hat{\boldsymbol{h}}_{j_{k_1}}^{l-k_1-1})$ in the empirical verification of Assumption 3.1 to estimate δ_p and take the estimation of neurons of $\boldsymbol{h}_{j_l}^0 \prod_{k_3=0}^{l-1} \boldsymbol{W}^{k_3}$ in the empirical verification of Assumption 3.1 to estimate $[h_i^l]_{p,t,\phi}$. In practice, we will obtain 100(message propagation paths) pairs of both δ_p and $[h_i^l]_{p,t,\phi}$. We then report the average and standard deviation of Pearson correlation between δ_p and $[h_i^l]_{p,t,\phi}$, between δ_p^2 and $[h_i^l]_{p,t,\phi}^2$, in Table 7.

In Table 7, the row titled with **Expected** indicates that we require the Pearson correlation to be 0 to hold Assumption 3.3. We observe that all average numbers of the Pearson correlation are less than or equal to 0.1.

Table 7. Estimation of the Pearson correlation between δ_p and $[h_i^l]_{p,t,\phi}$, between δ_p^2 and $[h_i^l]_{p,t,\phi}^2$. We present the the Pearson correlation between δ_p and $[h_i^l]_{p,t,\phi}$ in rows that have dataset names with subscripts 1, and the Pearson correlation between δ_p^2 and $[h_i^l]_{p,t,\phi}^2$ in rows that have dataset names with subscripts 2.

Dataset	Layers 1	2	3
cora$_1$	0.04±0.03	0.05±0.03	0.06±0.04
cora$_2$	0.05±0.03	0.04±0.03	0.03±0.03
pubmed$_1$	0.04±0.03	0.04±0.03	0.06±0.04
pubmed$_2$	0.05±0.04	0.04±0.03	0.03±0.03
citeseer$_1$	0.04±0.03	0.05±0.04	0.04±0.03
citeseer$_2$	0.04±0.05	0.05±0.03	0.05±0.03
arxiv$_1$	0.02±0.01	0.03±0.01	0.1±0.01
arxiv$_2$	0.03±0.01	0.04±0.01	0.01±0.01
Expected	0.0	0.0	0.0

Assumption 3.4 Assuming that δ_p is a Bernoulli random variable, it's known that mean of a Bernoulli random variable is equal to its success probability. With the same experiment setting as in the empirical verification of Assumption 3.3, we have 100(neurons) * 100(message propagation paths) observations of δ_p. We calculate the mean values and standard deviation of δ_p and report results in Table 8

In Table 8, the row titled with **Expected** indicates that we require the mean of δ_p to be 0.5 to hold Assumption 3.4. We observe that averages of δ_p are quite close to 0.5.

Table 8. Estimation of the success probability of δ_p.

Dataset	Layers		
	1	2	3
cora	0.47±0.03	0.24±0.02	0.13±0.01
pubmed	0.50±0.03	0.25±0.02	0.13±0.02
citeseer	0.52±0.03	0.27±0.02	0.14±0.02
arxiv	0.54±0.01	0.27±0.01	0.11±0.01
Expected	0.5	0.25	0.125

Theorem 3.5 Let's recap the forward propagation of the neuron t of the node i at the path p is presented as follows:

$$[h_i^l]_{p,t} = (\prod_{k_1=0}^{l-1} \delta(\hat{h}_{j_{k_1},t}^{l-k_1-1}))(\prod_{k_2=0}^{l-1} d_{j_{k_2}j_{k_2+1}})(\sum_{\substack{(t_0\cdots t_{l-1})\in\\ \hat{\mathbb{Z}}(l-1)}} h_{j_l,t_0}^0 \prod_{k_3=0}^{l-1} w_{t_{k_3},t_{k_3+1}}^k) \tag{15}$$

Let's denote $\prod_{k_1=0}^{l-1} \delta(\hat{h}_{j_{k_1},t}^{l-k_1-1})$ as δ_p, and $\prod_{k_2=0}^{l-1} d_{j_{k_2}j_{k_2+1}}$ as d_p, then we have:

$$var([h_i^l]_{p,t}) = d_p^2\ var(\delta_p \sum_{\substack{(t_0\cdots t_{l-1})\in\\ \hat{\mathbb{Z}}(l-1)}} h_{j_l,t_0}^0 \prod_{k=0}^{l-1} w_{t_k,t_{k+1}}^k) \tag{16}$$

Let's recap the expression of weight propagation path:

$$[h_i^l]_{p,t,\phi} = h_{j_l,t_0}^0 w_{t_0,t_1}^0 \cdots w_{t_{l-1},t}^{l-1} \tag{17}$$

It's obvious that $\sum_{\substack{(t_0\cdots t_{l-1})\in\\ \hat{\mathbb{Z}}(l-1)}} h_{j_l,t_0}^0 \prod_{k=0}^{l-1} w_{t_k,t_{k+1}}^k$ in (16) is the sum over $[h_i^l]_{p,t,\phi}$. We now derive the variance of $\delta_p[h_i^l]_{p,t,\phi}$:

$$var(\delta_p[h_i^l]_{p,t,\phi}) = (h_{j_l,t_0}^0)^2\ var(\delta_p \prod_{k=0}^{l-1} w_{t_k,t_{k+1}}^k) \tag{18}$$

where

$$var(\delta_p \cdot \prod_{k=0}^{l-1} w_{t_k,t_{k+1}}^k) = cov[\delta_p^2, \prod_{k=0}^{l-1} (w_{t_k,t_{k+1}}^k)^2] \tag{19a}$$

$$+ [var(\delta_p) + E^2(\delta_p)] \cdot [var(\prod_{k=0}^{l-1} w_{t_k,t_{k+1}}^k) + E^2(\prod_{k=0}^{l-1} w_{t_k,t_{k+1}}^k)] \tag{19b}$$

$$- [cov(\delta_p, \prod_{k=0}^{l-1} w_{t_k,t_{k+1}}^k) + E(\delta_p)E(\prod_{k=0}^{l-1} w_{t_k,t_{k+1}}^k)]^2 \tag{19c}$$

$w_{t_k,t_{k+1}}^k$ for different k are independent and have the same mean value 0, thus both $E(\prod_{k=0}^{l-1} w_{t_k,t_{k+1}}^k) = \prod_{k=0}^{l} E(w_{t_k,t_{k+1}}^k)$ and $E^2(\prod_{k=0}^{l-1} w_{t_k,t_{k+1}}^k)$ are equal to 0, $var(\prod_{k=0}^{l-1} w_{t_k,t_{k+1}}^k) = \prod_{k=0}^{l}(var(w_{t_k,t_{k+1}}^k)+E^2(w_{t_k,t_{k+1}}^k))-$

$\prod_{k=0}^{l}(E^2(w^k_{t_k,t_{k+1}}))$ is equal to $\prod_{k=0}^{l-1} var(w^k_{t_k,t_{k+1}})$. Additionally, given that Assumption 3.4 holds, δ_p is a Bernoulli random variable and has the success probability 0.5^l, where l is the length of p. Furthermore, when Assumption 3.3 holds, the correlation between δ_p and $\prod_{k=0}^{l-1} w^k_{t_k,t_{k+1}}$, between δ_p^2 and $\prod_{k=0}^{l-1}(w^k_{t_k,t_{k+1}})^2$, are both approximately equal to 0. As a result, we have:

$$var(\delta_p \prod_{k=0}^{l-1} w^k_{t_k,t_{k+1}}) = 0.5^l \cdot \prod_{k=0}^{l-1} var(w^k_{t_k,t_{k+1}}) \tag{20}$$

Combining Equations (18) and (20), we have:

$$var(\delta_p [h^l_i]_{p,t,\phi}) = (h^0_{j_l,t_0})^2 0.5^l \cdot \prod_{k=0}^{l-1} var(w^k_{t_k,t_{k+1}}) \tag{21}$$

When Assumption 3.2 holds, (16) is equal to:

$$\begin{aligned} var([h^l_i]_{p,t}) &= d_p^2 \sum_{\phi\in\Phi_{i,p,t}} var(\delta_p [h^l_i]_{p,t,\phi}) && \text{(22a)}\\ &= d_p^2\, 0.5^l \cdot \prod_{k=0}^{l-1} var(w^k_{t_k,t_{k+1}}) \cdot \sum_{\phi\in\Phi_{i,p,t}} (h^0_{j_l,t_0})^2 && \text{(22b)}\end{aligned}$$

$var(h^l_{i,t})$ is equal to the summation of $var([h^l_{i,t}]_{p,t})$ for all p in $\mathbb{P}_{i,l}$, thus we have:

$$\begin{aligned} var(h^l_{i,t}) &= var(\sum_{p\in\mathbb{P}_{i,l}} var([h^l_i]_{p,t})) && \text{(23a)}\\ &= \sum_{p\in\mathbb{P}_{i,l}} var([h^l_i]_{p,t}) + 2\sum_{i<j} cov([h^l_i]_{p_i,t}, [h^l_i]_{p_j,t}) && \text{(23b)}\end{aligned}$$

where

$$cov([h^l_i]_{p_i,t}, [h^l_i]_{p_j,t}) = d_{p_i} d_{p_j} cov(\sum_{\phi\in\Phi_{i,p,t}} \delta_p [h^l_i]_{p,t,\phi}, \sum_{\phi\in\Phi_{j,p,t}} \delta_p [h^l_j]_{p,t,\phi}) \tag{24}$$

where

$$\begin{aligned} &cov(\sum_{\phi\in\Phi_{i,p,t}} \delta_p [h^l_i]_{p,t,\phi}, \sum_{\phi\in\Phi_{j,p,t}} \delta_p [h^l_j]_{p,t,\phi}) && \text{(25a)}\\ &= cov(\sum_{\phi\in\Phi_{i,p,t}} \delta_p h^0_{j_l,t_0} \prod_{k=0}^{l-1} var(w^k_{t_k,t_{k+1}}), \sum_{\phi\in\Phi_{j,p,t}} \delta_{p_j} h^0_{j'_l,t_0} \prod_{k=0}^{l-1} var(w^k_{t_k,t_{k+1}})) && \text{(25b)}\\ &= h^0_{j_l,t_0} h^0_{j'_l,t_0} cov(\sum_{\phi\in\Phi_{i,p,t}} \delta_p \prod_{k=0}^{l-1} var(w^k_{t_k,t_{k+1}}), \sum_{\phi\in\Phi_{j,p,t}} \delta_{p_j} \prod_{k=0}^{l-1} var(w^k_{t_k,t_{k+1}})) && \text{(25c)}\end{aligned}$$

When Assumption 3.1 holds, the Pearson correlation between $\sum_\phi \delta_p \prod_{k=0}^{l-1} var(w^k_{t_k,t_{k+1}})$ and $\sum_\phi \delta_{p_j} \prod_{k=0}^{l-1} var(w^k_{t_k,t_{k+1}})$ is close to 1, we thus use $\sqrt{var(\sum_\phi \delta_p \prod_{k=0}^{l-1} var(w^k_{t_k,t_{k+1}}))var(\sum_\phi \delta_{p_j} \prod_{k=0}^{l-1} var(w^k_{t_k,t_{k+1}}))}$ to approximate the covariance term. And according to the analysis above, the correlation between different weight propagation paths is approximately 0, thus the covariance term can be approximated by

$\sqrt{\sum_{\phi} var(\delta_p \prod_{k=0}^{l-1} var(w_{t_k,t_{k+1}}^k)) \sum_{\phi} var(\delta_{p_j} \prod_{k=0}^{l-1} var(w_{t_k,t_{k+1}}^k))} = \sum_{\phi} var(\delta_p \prod_{k=0}^{l-1} var(w_{t_k,t_{k+1}}^k))$. There are $\prod_{l'=0}^{l-1} m_1^{(l')}$ weight propagation paths in $\Phi_{i,p,t}$ for L layers. We thus replace $\sum_{\phi} var(\delta_p \prod_{k=0}^{l-1} var(w_{t_k,t_{k+1}}^k))$ with $\prod_{l'=0}^{l-1} m_1^{(l')} \cdot var(\delta^l \prod_{k=0}^{l-1} var(w_{t_k,t_{k+1}}^k))$. As a result, with $\hat{h}_{j_l,t_0}^0$ to denote the mean over elements of $\boldsymbol{h}_{np_i}^0$ we have:

$$var(h_{i,t}^l) = \sum_{p \in \mathbb{P}_{i,l}} (\prod_{j,j' \in p} d_{jj'})^2 0.5^l \cdot \prod_{k=0}^{l-1} var(w_{t_k,t_{k+1}}^k) \sum_{\phi \in \Phi_{i,p,t}} (\hat{h}_{j_l,t_0}^0)^2 \tag{26}$$

$$+ 2m^l \cdot 0.5^l \cdot \prod_{k=0}^{l-1} var(w_{t_k,t_{k+1}}^k) \sum_{\substack{i<j \\ p_i,p_j \in \mathbb{P}_{i,l}}} (\prod_{t,t' \in p_i} d_{tt'} \prod_{t,t' \in p_j} d_{tt'})(\hat{h}_{j_l,t_0}^0 \hat{h}_{j_l',t_0}^0) \tag{27}$$

where

$$\sum_{p \in \mathbb{P}_{i,l}} (\prod_{j,j' \in p} d_{jj'})^2 \sum_{\phi \in \Phi_{i,p,t}} (\hat{h}_{j_l,t_0}^0)^2 = \prod_{l'=0}^{l-1} m_1^{(l')} \sum_{p \in \mathbb{P}_{i,l}} (\prod_{j,j' \in p} d_{jj'} \hat{h}_{j_l,t_0}^0)^2 \tag{28}$$

and

$$2 \sum_{\substack{i<j \\ p_i,p_j \in \mathbb{P}_{i,l}}} (\prod_{t,t' \in p_i} d_{tt'} \prod_{t,t' \in p_j} d_{tt'})(\hat{h}_{j_l,t_0}^0 \hat{h}_{j_l',t_0}^0) \tag{29}$$

$$= \sum_{\substack{i \neq j \\ p_i,p_j \in \mathbb{P}_{i,l}}} (\prod_{t,t' \in p_i} d_{tt'} \hat{h}_{j_l,t_0}^0)(\prod_{t,t' \in p_j} d_{tt'} \hat{h}_{j_l',t_0}^0) \tag{30}$$

$$= \sum_{p_i \in \mathbb{P}_{i,l}} (\prod_{t,t' \in p_i} d_{tt'} \hat{h}_{j_l,t_0}^0) \sum_{p_j \in \mathbb{P}_{i,l}} (\prod_{t,t' \in p_j} d_{tt'} \hat{h}_{j_l',t_0}^0) - \sum_{p_i' \in \mathbb{P}_{i,l}} (\prod_{t,t' \in p_i'} d_{tt'} \hat{h}_{j_l'',t_0}^0)^2 \tag{31}$$

Note that in Equation 31, $\sum_{p_i \in \mathbb{P}_{i,l}} (\prod_{t,t' \in p_i} d_{tt'} \hat{h}_{j_l,t_0}^0)$ is equal to $\sum_{p_j \in \mathbb{P}_{i,l}} (\prod_{t,t' \in p_j} d_{tt'} \hat{h}_{j_l',t_0}^0)$, and $\sum_{p_i' \in \mathbb{P}_{i,l}} (\prod_{t,t' \in p_i'} d_{tt'} \hat{h}_{j_l'',t_0}^0)^2$ of Equation 31 is equal to $\sum_{p \in \mathbb{P}_{i,l}} (\prod_{j,j' \in p} d_{jj'} \hat{h}_{j_l,t_0}^0)^2$ of Equation 28, thus for Equation 27 we have:

$$var(h_{i,t}^l) = \prod_{l'=0}^{l-1} m_1^{(l')} \cdot 0.5^l \cdot \prod_{k=0}^{l-1} var(w_{t_k,t_{k+1}}^k) \cdot \sum_{p \in \mathbb{P}_{i,l}} (\prod_{j,j' \in p} d_{jj'} \hat{h}_{j_l,t_0}^0)^2 \tag{32}$$

$$+ \prod_{l'=0}^{l-1} m_1^{(l')} \cdot 0.5^l \cdot \prod_{k=0}^{l-1} var(w_{t_k,t_{k+1}}^k) \cdot \sum_{p_i \in \mathbb{P}_{i,l}} (\prod_{t,t' \in p_i} d_{tt'} \hat{h}_{j_l,t_0}^0) \sum_{p_j \in \mathbb{P}_{i,l}} (\prod_{t,t' \in p_j} d_{tt'} \hat{h}_{j_l',t_0}^0) \tag{33}$$

$$- \prod_{l'=0}^{l-1} m_1^{(l')} \cdot 0.5^l \cdot \prod_{k=0}^{l-1} var(w_{t_k,t_{k+1}}^k) \cdot \sum_{p_i' \in \mathbb{P}_{i,l}} (\prod_{t,t' \in p_i'} d_{tt'} \hat{h}_{j_l'',t_0}^0)^2 \tag{34}$$

$$= \prod_{l'=0}^{l-1} m_1^{(l')} \cdot 0.5^l \cdot \prod_{k=0}^{l-1} var(w_{t_k,t_{k+1}}^k) \cdot [\sum_{p \in \mathbb{P}_{i,l}} (\prod_{t,t' \in p} d_{tt'} \hat{h}_{j_l'',t_0}^0)]^2 \tag{35}$$

$$= \prod_{l'=0}^{l-1} m_1^{(l')} \cdot 0.5^l \cdot \prod_{k=0}^{l-1} var(w_{t_k,t_{k+1}}^k) \cdot [\widetilde{\boldsymbol{A}}^l \boldsymbol{h}^0]_i^2 \tag{36}$$

where $[\boldsymbol{v}]_i^2$ denotes the i_{th} element of the element-wise square of the vector $\boldsymbol{v}$, $\widetilde{\boldsymbol{A}}$ is the renormalized adjacent matrix of the GCN, and $\boldsymbol{h}^0$ is a vector equal to $[h_0^0, h_1^0, \cdots, h_{|\mathbb{N}|-1}^0]^T$, where h_i^0 is the mean over elements of $\boldsymbol{h}_i^0$. Note that

$\widetilde{\boldsymbol{A}}^l \boldsymbol{h}^0$ is the message passing of the GCN with the input as a input graph and mean of node features. In practice, we use GCN implemented by DGL to accelerate computing $\widetilde{\boldsymbol{A}}^l \boldsymbol{h}^0$. Let $\boldsymbol{i} \in \mathbb{R}^{|\mathbb{N}|\times 1}$ be a vector with all ones, we intend to make $\sum_i var(h^l_{i,t})$ equal to $\sum_i var(h^{l+1}_{i,t})$. Thus we have:

$$\sum_i var(h^l_{i,t}) = \sum_i var(h^{l+1}_{i,t}) \tag{37}$$

$$\prod_{l'=0}^{l-1} m_1^{(l')} \cdot 0.5^l \cdot \prod_{k=0}^{l-1} var(w^k_{t_k,t_{k+1}}) \cdot \boldsymbol{i}^T [\widetilde{\boldsymbol{A}}^l \boldsymbol{h}^0]^2 = \prod_{l'=0}^{l} \cdot 0.5^{l+1} \cdot \prod_{k=0}^{l} var(w^k_{t_k,t_{k+1}}) \cdot \boldsymbol{i}^T [\widetilde{\boldsymbol{A}}^{l+1} \boldsymbol{h}^0]^2 \tag{38}$$

$$var(w^l) = \frac{2}{m_1^{(l)}} \frac{\boldsymbol{i}^T [\widetilde{\boldsymbol{A}}^{l-1} \boldsymbol{h}^0]^2}{\boldsymbol{i}^T [\widetilde{\boldsymbol{A}}^{l} \boldsymbol{h}^0]^2} \tag{39}$$

Assumption 3.7 We use the same experiment setting as in the empirical verification of Assumption 3.1, and calculate the mean and standard deviation of neurons of the softmax of h^L_i. In Table 9, we present evaluated results in the row titled with **Evaluated**. The row titled with **Expected** indicates that we require the mean of evaluated results to be expected numbers to hold Assumption 3.7. We observe that evaluated results are quite close to expected numbers.

Table 9. Estimation of the success probability of δ_p.

	Datasets			
	cora	pubmed	citeseer	arxiv
Evaluated	0.14±0.01	0.34±0.02	0.16±0.01	0.024±0.02
Expected	0.14	0.33	0.17	0.025

Proof of Theorem 2 Let's define the loss function as $Loss$, and L as the number of layers. The backward gradients of $Loss$ w.r.t $\boldsymbol{h}^l_i$ is $\frac{\partial Loss}{\partial \boldsymbol{h}^l_i}$, of which elements are assumed to be i.i.d. Thus we only consider the variance of $\frac{\partial Loss}{\partial h^l_{i,k}}$, where $h^l_{i,k}$ can be any element of $\boldsymbol{h}^l_i$. The variance of $\frac{\partial Loss}{\partial h^l_{i,k}}$ is:

$$\frac{\partial Loss}{\partial h^l_{i,k}} = \sum_{t\in\mathbb{N},k'} \frac{\partial Loss}{\partial h^L_{t,k'}} \frac{\partial h^L_{t,k'}}{\partial h^l_{i,k}} \tag{40}$$

where

$$\frac{\partial h^L_{t,k'}}{\partial h^l_{i,k}} = \sum_{p\in\mathbb{P}_{i^l,t^L}} \frac{\partial [h^L_{t,k'}]_p}{\partial h^l_{i,k}} \tag{41}$$

where $\mathbb{P}_{i^l,t^L}$ denote the set of all message propagation paths from $\boldsymbol{h}^l_i$ to $\boldsymbol{h}^L_t$. We firstly look at $\frac{\partial h^L_{t,k'}}{\partial h^l_{i,k}}$. $[h^L_{t,k'}]_p$ is equal to $\delta_p^{L-l} \prod_{j,j'\in p} d_{jj'} \sum_{k_{L-l}} \sum_{k_{L-2}} \cdots \sum_{k_{l+1}} h^l_{i,k} w^l_{k,k_{l+1}} w^{l+1}_{k_{l+1},k_{l+2}} \cdots w^{L-1}_{k_{L-1},k'}$, Thus we have:

$$\frac{\partial [h^L_{t,k'}]_p}{\partial h^l_{i,k}} = \delta_p^{L-l} \prod_{j,j'\in p} d_{jj'} \sum_{k_{L-l}} \sum_{k_{L-2}} \cdots \sum_{k_{l+1}} w^l_{k,k_{l+1}} w^{l+1}_{k_{l+1},k_{l+2}} \cdots w^{L-1}_{k_{L-1},k'} \tag{42}$$

and

$$var(\frac{\partial[h^L_{t,k'}]_p}{\partial h^l_{i,k}}) = m^{L-l-1}(\prod_{j,j'\in p} d_{jj'})^2\rho_{L-l}\cdot\prod_{k=l+1}^{L-1} var(w^k_{t_k,t_{k+1}}) \tag{43}$$

where the m above is the *fan_out* of each layer, which is different from the one defined in in proof of Theorem 1. Similar to Equation 27 and its subsequent derivations, we have:

$$var(\frac{\partial h^L_{t,k'}}{\partial h^l_{i,k}}) = m^{L-l-1}\cdot\rho_{L-l}\cdot\prod_{k=l+1}^{L-1} var(w^k_{t_k,t_{k+1}})\cdot \boldsymbol{i}^T[\widetilde{\boldsymbol{A}}^{L-l}\boldsymbol{i}]^2 \tag{44}$$

Assuming that the loss function is cross entropy, and the label of node t is $y(t)$, we have:

$$\frac{\partial Loss}{\partial h^L_{t,k'}} = \begin{cases} \frac{s(\boldsymbol{h}^L_t)_{k'}-1}{|\mathbb{N}|} & k' = y(t) \\ \frac{s(\boldsymbol{h}^L_t)_{k'}}{|\mathbb{N}|} & k' \neq y(t) \end{cases} \tag{45}$$

where $s(\boldsymbol{v})_i$ is the i_{th} element of the *softmax* of a vector $\boldsymbol{v}$. **Experiments** show that at the first epoch, $s(\boldsymbol{h}^L_t)_{k'}$ is approximately $1/C$, where C is the number of classes. Thus we have:

$$var(\frac{\partial Loss}{\partial h^l_{i,k}}) = var(\sum_{t\in\mathbb{N},k'} \frac{\partial Loss}{\partial h^L_{t,k'}}\frac{\partial h^L_{t,k'}}{\partial h^l_{i,k}}) \tag{46}$$

$$= \sum_{t\in\mathbb{N},k'} (\frac{\partial Loss}{\partial h^L_{t,k'}})^2 var(\frac{\partial h^L_{t,k'}}{\partial h^l_{i,k}}) \tag{47}$$

$$= \frac{1}{|\mathbb{N}|^2}(1-\frac{1}{C})\cdot m^{L-l-1}\cdot\rho_{L-l}\cdot\prod_{k=l+1}^{L-1} var(w^k_{t_k,t_{k+1}})\cdot[\widetilde{\boldsymbol{A}}^{L-l}\boldsymbol{i}]^2_i \tag{48}$$

Similar to the proof of Theorem 1, we intend to make $\sum_i var(\frac{\partial Loss}{\partial h^l_{i,k}})$ equal to $\sum_i var(\frac{\partial Loss}{\partial h^{l+1}_{i,k}})$, thus we have:

$$\frac{1}{|\mathbb{N}|^2}(1-\frac{1}{C})\cdot m^{L-l-1}\cdot\rho_{L-l}\cdot\prod_{k=l+1}^{L-1} var(w^k_{t_k,t_{k+1}})\cdot \boldsymbol{i}^T[\widetilde{\boldsymbol{A}}^{L-l}\boldsymbol{i}]^2 \tag{49}$$

$$=\frac{1}{|\mathbb{N}|^2}(1-\frac{1}{C})\cdot m^{L-l-2}\cdot\rho_{L-l-1}\cdot\prod_{k=l+2}^{L-1} var(w^k_{t_k,t_{k+1}})\cdot \boldsymbol{i}^T[\widetilde{\boldsymbol{A}}^{L-l-1}\boldsymbol{i}]^2 \tag{50}$$

Then we have:

$$var(w^k_{t_k,t_{k+1}}) = \frac{2}{m^k_2}\frac{\boldsymbol{i}^T[\widetilde{\boldsymbol{A}}^{L-l-1}\boldsymbol{i}]^2}{\boldsymbol{i}^T[\widetilde{\boldsymbol{A}}^{L-l}\boldsymbol{i}]^2} \tag{51}$$

For the last layer $var(w^{L-1})$, we obtain it by making $\sum_i var(\frac{\partial Loss}{\partial h^L_{i,k}})$ equal to $\sum_i var(\frac{\partial Loss}{\partial h^{L-1}_{i,k}})$. According to Equation 45, the $var(\frac{\partial Loss}{\partial h^L_{i,k}})$ is approximately equal to $1/(|\mathbb{N}|^2 C)$. Thus we have:

$$\frac{|\mathbb{N}|}{|\mathbb{N}|^2 C} = \frac{1}{|\mathbb{N}|^2}(1-\frac{1}{C}) \cdot var(w^{L-1}) \cdot \boldsymbol{i}^T[\widetilde{\boldsymbol{A}}\boldsymbol{i}]^2 \tag{52}$$

$$var(w^{L-1}) = \frac{|\mathbb{N}|}{(C-1)\boldsymbol{i}^T[\widetilde{\boldsymbol{A}}\boldsymbol{i}]^2} \tag{53}$$

B. Experimental Setting

In this section, we present design space of hyperparameter tunning used in Section 5.

B.1. Node Classification

For node classification tasks, we perform a grid search to tune the hyperparameters. We list the hyperparameter tunning settings in Table 10, where lr and wd denotes the learning rate and the weight decay, respectively, and num_layers represents the number of GNN layers. The patience are used for early-stopping.

Table 10. The hyperparameter tunning setting of experiments in Table 2.

Datasets	Hyper-parameters
cora	hidden_channels: {32, 64, 128}, num_layers: {2, 3, 4}, lr: {1e-3, 5e-3, 1e-2, 5e-2}, epochs: 1000, patience: 20, dropout: {0.0, 0.5}, wd: {0.0, 5e-6}
pubmed	hidden_channels: {32, 64, 128}, num_layers: {2, 3, 4}, lr: {1e-3, 5e-3, 1e-2, 5e-2}, epochs: 1000, patience: 20, dropout: {0.0, 0.5}, wd: {0.0, 5e-6}
citeseer	hidden_channels: {32, 64, 128}, num_layers: {2, 3, 4}, lr: {1e-3, 5e-3, 1e-2, 5e-2}, epochs: 1000, patience: 20, dropout: {0.0, 0.5}, wd: {0.0, 5e-6}
arxiv	hidden_channels: {128, 256}, num_layers: {3, 4, 5}, lr: {1e-3, 5e-3, 1e-2, 5e-2}, epochs: 1000, patience: 100, wd: {0.0, 5e-6, 5e-5}, dropout: {0.0, 0.5}
proteins	hidden_channels: {128, 256}, num_layers: 3, lr: {1e-3, 5e-3, 1e-2, 5e-2}, dropout: {0.0, 0.5}, wd: {0.0, 5e-6}, epochs: 2000
products	hidden_channels: 256, num_layers: {3, 4}, lr: {1e-3, 5e-3}, dropout: 0.5, wd: {0.0, 5e-5}, epochs: 100

B.2. Link Prediction

For all OGB datasets, we follow the rules of OGB and employ the same data splitting. We use the Adam optimizer with zero weight_decay. We list the hyperparameter tunning settings for the link prediction tasks in Table 11, where lr represents the learning rate and num_layers represents the number of GNN layers. For NGNN models, we use ngnn_type to denote the position where the non-linear layers are inserted, (for instance, *input* means applying NGNN to only the input GNN layer), and use num_ngnn_layers to denote the number of nonlinear layers in each NGNN block. Since the citation2 datasets is too large, we use the memory-friendly variants of (NGNN-)GCN and (NGNN-)GraphSAGE, namely ClusterGCN and Neighbor Sampling(SAGE aggregation).

B.3. Graph Classification

For graph classification tasks, the grid search settings for hyperparameter tunning are listed in Table 12, where lr and wd denotes the learning rate and the weight decay, respectively, and num_layers represents the number of GNN layers. For NGNN variants, ngnn_type refers to the position of the linear layer. For example, *input* means there is one linear layer stacked on the first GNN layer, and *all* means each GNN layer is equipped with a linear layer. num_ngnn_layers refers to the number of linear layers in the GNN model. Specific details can be found in (Song et al., 2021).

Table 11. The hyperparameter tunning setting of experiments in Table 3.

Datasets	Models	Hyper-parameters
ddi	GCN	lr: {0.002, 0.005}, batch_size: {16384, 32768}, dropout: {0.4, 0.5, 0.6}, epoch: {1600, 2400}, num_layers: 2, hidden_channels: 256
	NGNN-GCN	ngnn_type: {input, all}, num_ngnn_layers: 1, lr: {0.001, 0.0015, 0.002, 0.005}, batch_size: {8192, 16384, 32768}, dropout: {0.2, 0.3, 0.4, 0.5, 0.6, 0.7}, epoch: {800, 1600, 2400}, num_layers: 2, hidden_channels: 256
	GraphSAGE	lr: {0.001, 0.002}, batch_size: 32768, dropout: {0.1, 0.2, 0.3, 0.4}, epoch: {800, 1200, 2000}, num_layers: 2, hidden_channels: 256
	NGNN-GraphSAGE	ngnn_type: input, num_ngnn_layers: 1, lr: {0.0005, 0.001, 0.005}, batch_size: {8192, 16384, 32768}, dropout: {0, 0.1, 0.2, 0.3, 0.4}, epoch: {600, 1200, 2000}, num_layers: 2, hidden_channels: 256
collab	GCN	lr: {0.0005, 0.001}, batch_size: {32768, 65536}, dropout: {0.2, 0.3}, epoch: {800, 1200}, num_layers: {3, 4}, hidden_channels:256
	NGNN-GCN	ngnn_type: {input, hidden, all}, num_ngnn_layers: 2, lr: {0.0005, 0.001, 0.002, 0.005}, batch_size: {32768, 65536}, dropout: {0.2, 0.3}, epoch: {800, 1200}, num_layers: {3, 4}, hidden_channels:256
	GraphSAGE	lr: {0.0005, 0.001}, batch_size: {65536, 131072}, dropout: 0.2, epoch: {600, 900}, num_layers: 4, hidden_channels: 256
	NGNN-GraphSAGE	ngnn_type: {input, hidden, all}, num_ngnn_layers: 2, lr: {0.0005, 0.001, 0.002}, batch_size: {32768, 65536, 131072}, dropout: {0, 0.1, 0.2, 0.3}, epoch: {400, 800, 1200}, num_layers: {3, 4}, hidden_channels: 256
citation2	GCN	lr: {0.0003, 0.0005, 0.001}, batch_size: 256, dropout: {0, 0.1, 0.2}, epoch: 200, num_layers: 3, hidden_channels: 256
	NGNN-GCN	ngnn_type: {hidden, input}, num_ngnn_layers: {1, 2}, lr: {0.0003, 0.0005}, batch_size: 256, dropout: {0, 0.1}, epoch: 200, eval_step: 10, num_layers: 3, hidden_channels: 256
	GraphSAGE	lr: {0.0003, 0.0005, 0.001}, batch_size: 1024, dropout: {0, 0.2}, epoch: 200, num_layers: 3, hidden_channels:256
	NGNN-GraphSAGE	ngnn_type: {hidden, input}, num_ngnn_layers: {1, 2}, lr: {0.0003, 0.0005, 0.001}, batch_size: 1024, dropout: {0, 0.2}, epoch: 200, num_layers: 3, hidden_channels: 256
ppa	GCN	lr: 0.001, batch_size: {32768, 49152, 65536}, dropout: {0.2, 0.3}, epoch: {120, 150}, num_layers: {3, 4}, hidden_channels: 256
	NGNN-GCN	ngnn_type: input, num_ngnn_layers: 2, lr: 0.001, batch_size: {32768, 49152, 65536}, dropout: {0.2, 0.3}, epoch: {120, 150}, num_layers: {3, 4}, hidden_channels: 256
	GraphSAGE	lr: {0.001, 0.0015}, batch_size: {49152, 65536}, dropout: 0.2, epoch: {120, 150}, num_layers: 4, hidden_channels: 256
	NGNN-GraphSAGE	ngnn_type: input, num_ngnn_layers: 2, lr: {0.001, 0.0015}, batch_size: {49152, 65536}, dropout: 0.2, epoch: {120, 150}, num_layers: {3, 4}, hidden_channels: 256

Table 12. The hyperparameter tunning setting of experiments in Table 4.

Datasets	Hyper-parameters
molhiv	lr: {5e-4, 1e-4, 5e-3, 1e-3}, dropout: {0.0, 0.5}, wd: {0.0, 5e-6}, batch_size: 32, hidden_channels: 300, num_layer: 5
molpcba	lr: {5e-4, 1e-4, 5e-3, 1e-3}, dropout: {0.0, 0.5}, wd: {0.0, 5e-6}, batch_size: 32, hidden_channels: 300, num_layer: 5

C. Miscellaneous

We present intermediate steps to derive (3) in following equations.

$$\begin{aligned}
\boldsymbol{h}_i^l =&\delta(\hat{\boldsymbol{h}}_i^{l-1}) \odot \hat{\boldsymbol{h}}_i^{l-1} \\
=&\delta(\hat{\boldsymbol{h}}_i^{l-1}) \odot \sum_{j_1 \in \mathbb{N}(i)} d_{ij_1} \delta(\hat{\boldsymbol{h}}_{j_1}^{l-1}) \odot \hat{\boldsymbol{h}}_{j_1}^{l-1} \cdot \boldsymbol{W}^{l-1} \\
=&\delta(\hat{\boldsymbol{h}}_i^{l-1}) \odot \sum_{j_1 \in \mathbb{N}(i)} d_{ij_1} \delta(\hat{\boldsymbol{h}}_{j_1}^{l-1}) \odot \sum_{j_2 \in \mathbb{N}(j_1)} d_{j_1 j_2} \delta(\hat{\boldsymbol{h}}_{j_2}^{l-2}) \\
&\odot \hat{\boldsymbol{h}}_{j_2}^{l-2} \cdot \boldsymbol{W}^{l-2} \boldsymbol{W}^{l-1} \\
&\dots \\
=&\delta(\hat{\boldsymbol{h}}_i^{l-1}) \odot (\prod_{\substack{k_1=0 \\ \odot}}^{l-2} \sum_{\substack{j_{k_1+1} \in \\ \mathbb{N}(j_{k_1})}} d_{j_{k_1} j_{k_1+1}} \delta(\hat{\boldsymbol{h}}_{j_{k_1+1}}^{l-k_1-2})) \\
&\odot (\sum_{j_l \in \mathbb{N}(j_{l-1})} d_{j_{l-1} j_l} \boldsymbol{h}_{j_l}^0) \cdot (\prod_{k_2=0}^{l-1} \boldsymbol{W}^{k_2})
\end{aligned} \tag{54}$$