DRGCN: Learning Dynamic Initial Residual via Evolving for Deep Graph Convolutional Networks

Anonymous Author(s)

ABSTRACT

Graph convolutional networks (GCNs) have been proved to be very practical to handle various graph-related tasks. It has attracted considerable research interest to study the deep GCNs, due to their potential superior performance compared to the shallow ones. However, simply increasing network depth will, on the contrary, hurt the performance due to the over-smoothing problem. Adding residual connection is proved to be effective for learning deep convolutional neural networks (deep CNNs), it is not trivial while applying to deep GCNs. Recent works proposed an initial residual mechanism that did alleviate the over-smoothing problem in deep GCNs. However, according to our study, their algorithms are quite sensitive to different datasets since they used a fixed residual weight (we call it static residual).In their setting, the differences (dynamic) and relations (evolving) of how residual applies in each layer are ignored, which oversimplified the model and cause the residual weight hard to tune in some datasets. To this end, we propose a novel model called **D**ynamic initial **R**esidual **G**raph **C**onvolutional **N**etwork (DRGCN)[1]. Firstly, we use a dynamic initial residual block in each layer to adaptively fetch information from the initial representation. Secondly, we use a recurrent module to model the residual evolving pattern between layers. Our experimental results show that our model effectively relieves the problem of over-smoothing in deep GCNs and outperforms the state-of-the-art methods on various benchmark datasets. Moreover, we develop a mini-batch version of DRGCN which can be applied to large-scale data. Coupling with several fair training techniques, our model reaches new SOTA results on the large-scale *ogbn-arxiv* dataset of Open Graph Benchmark (OGB)[2].

CCS CONCEPTS

• **Theory of computation → Graph algorithms analysis**.

KEYWORDS

graph representation learning; graph convolutional network; over-smoothing; dynamic residual; initial residual

ACM Reference Format:

Anonymous Author(s). 2022. DRGCN: Learning Dynamic Initial Residual via Evolving for Deep Graph Convolutional Networks. In *Proceedings of In Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22).* ACM, New York, NY, USA, 11 pages. https://doi.org/10.XXXX/XXXXXXX.XXXXXXX

[1]Codes are available at https://github.com/anonymousaabc/DRGCN

[2]https://ogb.stanford.edu/

KDD '22, August 14-18, 2022, Washington, America

ACM ISBN 978-1-4503-XXXX-X/22/08…$15.00
https://doi.org/10.XXXX/XXXXXXX.XXXXXXX

1 INTRODUCTION

Graph representation learning is the main task for graph-structured data. Graph Convolution Networks (GCNs) [3, 7, 11, 18, 49] and its attentional variants (*e.g.*, GAT) [35–37, 39, 46] have been shown to be quite successful especially on graph representation learning tasks. These models are powerful on node classification [18, 25, 42], link prediction [2, 28, 50] and clustering [5] tasks, because their remarkable ability to iteratively aggregate information from graph neighborhoods. GCNs also achieve great success on applications like recommender system [12, 14, 44] and traffic forecasting [10, 45, 46]. Motivated by deep neural networks from other areas such as computer vision [13, 40], where commonly deeper models have superior performance than shallow ones, researchers have made great progress on the design of deeper GCN structures [19, 20, 22, 30, 33, 41, 43]. Intuitively, the deep version of GCNs has the ability to aggregate knowledge from remote hops of neighbors for target nodes. But it fails because these models end up with severe over-smoothing [21, 24] issue, which shows that after too many rounds of the aggregation process, the representations of nodes converge to the same pattern and become indistinguishable.

Recently, several research efforts [21, 30, 48] have been devoted to tackling the over-smoothing issue. Data augmentation strategies [8, 27, 30, 38] can be used as a general technique for various graph convolution networks. They are effective for the over-fitting problem, but only alleviate the over-smoothing issue to some extent. DAGNN [26] decouples transformation and propagation operations and adaptively incorporating information from large receptive fields. GRAND [9] uses DropNode as a random perturbation method and consistency regularization to enforce the consistency of each node's classification confidence.

Besides the works mentioned above, the residual connection is another powerful strategy to solve the over-smoothing issue. JKNet [43] employs dense skip connections to combine information of all layers for final node representation. APPNP [19] proposes initial residual in the context of Personalized PageRank [29] to constructs a skip connection to the initial representation. However, these residual-involved models achieve their best results when the model is shallow, the performance drops when networks become deeper. GCNII [4] goes further with the initial residual by introducing identity mapping, and alleviates the over-smoothing issue while achieving new state-of-the-art results on various tasks.

However, all of the previous works using residual connection treat residual weight as a static hyperparameter that needs tuning and is quite sensitive to datasets (Figure 1, left), the scale of training datasets size (Figure 1, middle), and model depths (Figure 1, right). It's a time-consuming process searching for the best hyperparameter. For more detailed experimental verification results

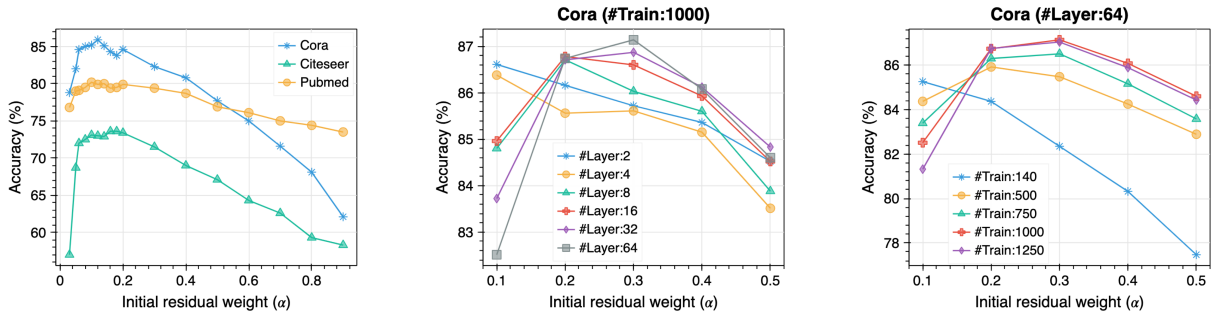

Figure 1: The sensitivity analysis results of static alpha for different datasets (left), different model depths (middle) and different training size (right).

corresponding to Figure 1, please refer to Table 10 and Table 11 in the appendix.

Intuitively, because nodes have different initial representations and neighbors, personalizing residual weight for each node makes sense. Moreover, shallow layers still memory the information from the initial representation which means we do not need to fetch much information by initial residual for them. On the other hand, as the layer goes deeper, we should fetch more information from the initial representation to prevent noise. And we believe that initial residual weights from shallow to deeper layers should obey some evolving pattern.

Accordingly, we argue that a powerful residual GCN model should have the ability to learn the dynamic residual weights automatically based on initial and current layer representation, and uncover the information evolving trend from shallow to deep layers. So, in this paper, we introduce DRGCN, a novel dynamic residual GCN model. Firstly, we use a dynamic initial residual block (DIRB) in each layer to adaptively fetch information from the initial representation. Secondly, we frame the residual weight dependency as a sequential evolving problem from shallow to deep layers and use a recurrent module to model the residual evolving pattern between layers. We compare DRGCN with other residual GCN models like GCNII on various datasets. The experimental results show that DRGCN not only alleviates the over-smoothing issue, but also outperforms the state-of-the-art (SOTA) models, especially in larger datasets. Additionally, the visualization of the residual evolving process demonstrates our assumption about the shallow and deep layers should be treated differently.

To summarize, the main contributions of our work are:

- We propose DRGCN, a model joint learning dynamic initial residual with the evolving mechanism. To the best of our knowledge, this is the first work introducing dynamic residual into graph convolution networks to relieve the over-smoothing issue.
- We conduct extensive experiments to demonstrate the consistent state-of-the-art performance of DRGCN on various benchmark datasets. Meanwhile, DRGCN shows good self-adaption to different datasets, model depths, and scale of data size without manual hyperparameter tuning.
- Our model reaches new SOTA results on the large-scale ogbn-arxiv dataset of the OGB leaderboard [16] at the time of submission, which demonstrates the generalization of our model on the large-scale dataset. We also introduce mini-batch DRGCN which optimize the memory consumption without degrading the performance.
- We visually analyze that information from shallow (short-distance) and deep (long-distance) layers should be learned individually, and the study of dynamic residual connection has great potential for graph representation learning.

2 PRELIMINARIES AND RELATED WORK

In this section, we first define some essential notations used in this paper. Given a connected undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with n nodes and e edges, where $v_i \in \mathcal{V}$ and $(v_i, v_j) \in \mathcal{E}$ denote node with index $i \in (1, 2, ..., n)$ and edge between node v_i and v_j respectively. We use $\mathbf{X} \in \mathbb{R}^{n \times d}$ to denote node feature matrix where d is the feature dimension of node. The topology information is described by the adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, and $\mathbf{A}_{ij} = 1$ if $(v_i, v_j) \in \mathcal{E}$, otherwise 0. For undirected graph, $\mathbf{A}$ is a symmetric matrix. Let $\mathbf{D}$ denote the diagonal degree matrix, where $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$ meaning the number of neighbors of v_i. When adding self-loop edges, the adjacency matrix and diagonal degree matrix become $\widetilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ and $\widetilde{\mathbf{D}} = \mathbf{D} + \mathbf{I}$ respectively, with $\mathbf{I}$ being the identity matrix.

2.1 Deep GCNs

Recent researches mainly focus on two types of GCNs, spectral-based [3, 4, 18, 21] and spatial-based [11, 32, 44]. For spectral-based manner, the vanilla GCN with two layers is expressed as

$$\mathbf{Z} = \widetilde{\mathbf{P}}\,\mathrm{ReLU}\left(\widetilde{\mathbf{P}}\mathbf{X}\mathbf{W}^{(0)}\right)\mathbf{W}^{(1)}. \tag{1}$$

where $\mathbf{Z} \in \mathbb{R}^{n \times c}$ is the output embedding matrix of target nodes and c is the number of classes and $\widetilde{\mathbf{P}} = \widetilde{\mathbf{D}}^{-1/2}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-1/2}$. In addition, $\mathbf{W}^{(0)} \in \mathbb{R}^{d \times d_h}$ and $\mathbf{W}^{(1)} \in \mathbb{R}^{d_h \times c}$ are weight matrices to map features.

To further improve the graph model performance, recent works focused on designing a deep structure for GCN (deep GCNs) [1, 19, 21, 24, 30, 30, 33]. Though from a different perspective and using various tricks, all of them are building from the same thoughts. Iteratively applying first-order graph convolution described in Eq.

1 vertically actually achieves the similar result as horizontally implementing L-localized convolutions, all of which aggregate information from the $(L-1)$-order neighborhood of target notes. Here L denotes the model depth, also the number of convolutional layers. In general, most of the deep GCNs adopt the following rule

$$\mathbf{H}^{(\ell+1)} = \sigma\left(\widetilde{\mathbf{P}}\mathbf{H}^{(\ell)}\mathbf{W}^{(\ell)}\right), \tag{2}$$

where $\sigma(\cdot)$ denotes the nonlinear activation function such as ReLU. Meanwhile, $\mathbf{H}^{(\ell)}$ and $\mathbf{W}^{(\ell)}$, $\ell = 0, \ldots, L-1$, are the input hidden node representation and weight matrix in the ℓ-th layer with $\mathbf{H}^{(0)} = \mathbf{X}$, $\mathbf{H}^{(\ell+1)}$ is the output node representation. So for each graph convolution layer, the hidden node representations are updated through feature propagation, linear transformation and a pointwise nonlinear activation.

To improve the scalability and performance of deep GCNs, many sampling-based models have been proposed such as GraphSAGE [11] and FastGCN [3]. Also, another group of work try to design powerful regularization methods like GRAND [9], whose core module DropNode can be regarded as general graph augmentation technique. Though effective for the over-fitting issue, these models still exist the over-smoothing issue especially for a large model depth and they achieve their best results with shallow layers.

DAGNN [26] proposed smoothness metric and theoretically make assumptions that the over-smoothing issue only happens when node representations propagate repeatedly for a large number of iterations, especially for a graph with sparsely connected edges. It demonstrated that disentanglement of transformation and propagation process in deep GCNs improved the model performance. In DAGNN, the transformation stage

$$\mathbf{H}^{(0)} = \mathrm{MLP}(\mathbf{X}), \tag{3}$$

and the propagation stage described as

$$\mathbf{H}^{(\ell)} = \widetilde{\mathbf{P}}^{\ell}\mathbf{H}^{(0)}, \tag{4}$$

where $\widetilde{\mathbf{P}}^{\ell}$ is the ℓ-th power of $\widetilde{\mathbf{P}}$.

2.2 Deep Residual GCNs

Residual methods have been demonstrated to be a simple but effective strategy to efficiently train very deep CNN and improve their performance in the Computer Vision research area [13, 34]. Motivated by this, another line of deep GCNs resort to residual methods to relieve the over-smoothing issue.

ResGCNs [21] propose a dense residual connection that combines the smoothed representation $\widetilde{\mathbf{P}}\mathbf{H}^{(\ell)}$ with $\mathbf{H}^{(\ell)}$. Consequently, the output node representation for layer ℓ is

$$\mathbf{H}^{(\ell+1)} = (1-\alpha)\widetilde{\mathbf{P}}\mathbf{H}^{(\ell)} + \alpha\mathbf{H}^{(\ell)}, \tag{5}$$

where α is the residual weight for all nodes and layers. Unfortunately, deep GCNs with dense residual connections merely relieves the over-smoothing problem.

Recently, APPNP [19] and GCNII [4] propose the initial residual connections which combine $\mathbf{H}^{(\ell)}$ with the initial representation $\mathbf{H}^{(0)}$ rather than the node representation from previous layer as descripted in Eq. 5. The initial residual mechanism is able to propagate information from the initial layer whose representation is proved to be crucial for nodes classification [26]. Formally, a deep graph convolution network with initial residual aggregation is defined as

$$\mathbf{H}^{(\ell+1)} = (1-\alpha)\widetilde{\mathbf{P}}\mathbf{H}^{(\ell)} + \alpha\mathbf{H}^{(0)}, \tag{6}$$

where α is the hyperparameter, and the initial node representation $\mathbf{H}^{(0)}$ is obtained by $\mathbf{X}$ using Eq. 3. Therefore, the initial residual connection ensures that the representation of each node in each layer retains at least α fraction information from the initial representation even if we stack large layers. Compared with dense residual connection, the initial residual connection effectively relieves over-smoothing issue as demonstrated in GCNII [4] when coupling with identity mapping strategy.

However, the initial residual in the previous work is static. Therefore, the initial residual weight is fixed for all nodes and layers, which needs a time-consuming hyperparameter tuning step. To amend the deficiency of static residual algorithms, we propose a novel dynamic residual model DRGCN. In this model, we introduce the dynamic initial residual block and model the residual evolving pattern using the recurrent module at the same time.

Expect from purely model design, another line of researches [22, 23] focused on how to optimize the GPU memory consumption of deep GCNs in large-scale scenario. RevGCN[22] studied several techniques such as weight tying and implicit differentiation to optimize the training efficiency of GNNs. RevGCN proposed reversible residual connections to increase the GPU memory efficiency and thus is able to train GNN with more than 1000 layers. Also, its variants showed competitive results on large-scale graph datasets. We will compare our model with RevGCN in large-scale scenario in Section 4.7.

3 DRGCN MODEL

3.1 Overall

In this section, we present the overall architecture of the DRGCN model. The proposed DRGCN framework is shown in Figure 2.

We first apply transformation as Eq. 3 on $\mathbf{X}$ to get the initial representation $\mathbf{H}^{(0)}$ before propagation process. In propagation phase, we design a dynamic and node-personalized DIRB, which use $\mathbf{H}^{(0)}$ and $\mathbf{H}^{(\ell)}$ as input to measure the similarity information between them.

However, solely using DIRB inevitably ignore the residual evolving pattern or dependencies between layers as the model goes deeper. Also, the initial residual weight in each layer should somehow memory the historical residual information. Therefore, we utilize a recurrent neural network (RNN) to model this initial residual evolving pattern while using DIRB's output as its input. As is shown at the top left of Figure 2. By coupling these two core components, we finally get the dynamic initial residual weight for every layer. The bottom left of Figure 2 shows the feature propagation process through graph convolution and Soft Gated Unit (SGU). For generality, we use GCNConv to represent the graph convolution module that aggregates messages from neighbors. Afterward, the result of GCNConv, initial representation, and dynamic initial

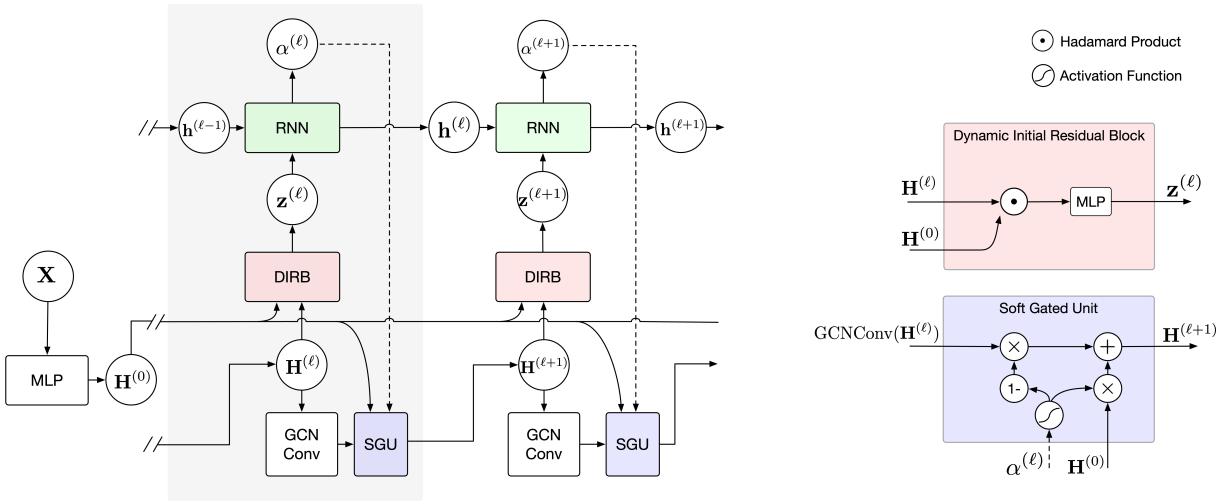

Figure 2: The overall architecture of DRGCN.

residual weight is passed to SGU and produces the final hidden representation of nodes in a weighted sum manner.

In summary, DRGCN is composed of two key components: 1) DIRB, which independently measures the similarity between initial representation and hidden representation, also dynamically determine the initial residual of each layer; 2) RNN module, which models initial residual evolving process sequentially and adaptively determine the weight of the initial representation in each layer.

We also propose a variant DRGCN* that additionally utilizes data augmentation and consistency regularization training skills. DRGCN* is more robust on datasets of different scales. We will describe the details of DRGCN* in section 3.5. In the following sections, we will introduce the mathematical expressions of our model in detail.

3.2 Dynamic Initial Residual Block

The top right of Figure 2 depicts the details of DIRB. Formally, The mathematical expression is defined as

$$\mathbf{z}^{(\ell)} = \mathrm{MLP}\left(\Phi\left(\widetilde{\mathbf{H}}^{(0)}, \widetilde{\mathbf{H}}^{(\ell)}\right)\right), \tag{7}$$

where Φ denotes the combination function, $\widetilde{\mathbf{H}}^{(0)} \in \mathbb{R}^{n\times d}$ is obtained by L2 regularization from the first layer representation $\mathbf{H}^{(0)}$, d is the hidden dimension size, n is the number of nodes in the graph. Similarly, $\widetilde{\mathbf{H}}^{(\ell)} \in \mathbb{R}^{n\times d}$ is the L2 regularization of the ℓ-th layer representation $\mathbf{H}^{(\ell)}$. The combination function generally uses Hadamard product operation. Nevertheless, *Sub* and *Concat* can also be used as Φ.

Theoretically, MLP can approximate any measurable function [15]. Therefore, for the purpose of modeling the similarity between $\mathbf{H}^{(0)}$ and $\mathbf{H}^{(\ell)}$, we use MLP to map the vector into residual weight scalar for each node. Hence, $\mathbf{z}^{(\ell)} \in \mathbb{R}^{n\times 1}$.

3.3 Modeling Residual Evolving Process

DIRB only takes initial representation and hidden representation of the current layer as input and doesn't take residual dependencies between layers into account. Therefore, $\mathbf{z}^{(\ell)}$ is not the final dynamic initial residual weight. We treat the residual dependencies modeling as a sequential evolving problem from shallow to deep layers, thus we apply RNN to model the residual evolving process as

$$\alpha^{(\ell)}, \mathbf{h}^{(\ell)} = \mathrm{RNN}\left(\mathbf{z}^{(\ell)}, \mathbf{h}^{(\ell-1)}\right), \tag{8}$$

$\mathbf{h}^{(\ell)} \in \mathbb{R}^{n\times 1}$ is the hidden state output of the ℓ-th layer. Specially, $\mathbf{h}^{(0)}$ is initialized from $\mathcal{U}(-\sqrt{k}, \sqrt{k})$ where $k = 1/d$. $\alpha^{(\ell)} \in \mathbb{R}^{n\times 1}$ is output of the ℓ-th layer of the recurrent neural network and also the final dynamic residual weight. $\alpha^{(\ell)}$ measures how much information of the initial representation should be retained in each propagation layer.

3.4 Propagation and Soft Gated Unit

In propagation phase, we use the spectral convolution described in Eq. 2 expect that we use $\mathbf{W}^{(\ell)}$ only for the last layer. And the propagation for aggregating messages in each layer is $\widetilde{\mathbf{P}}\mathbf{H}^{(\ell)}$. Remarkably, we can also use attentive based propagation methods, which forms DRGAT for large-scale datasets, see section 4.7.

To narrow the value range of the dynamic initial residual weight, we then utilize an activation function to $\alpha^{(\ell)}$ as below

$$\alpha^{(\ell)} = \sigma\left(\alpha^{(\ell)}\right), \tag{9}$$

where σ is the activation function and we use *Sigmoid* in our model. The final $\alpha^{(\ell)}$ is layer-dependent and personalized.

As is shown at the bottom right of Figure 2. For the last component SGU, we simply use a weight sum operation to combine the initial representation and hidden representation. Formally, the $(\ell+1)$-th layer output of the SGU module is defined as

$$\mathbf{H}^{(\ell+1)} = \mathrm{ReLU}\left(\left(1-\alpha^{(\ell)}\right)\widetilde{\mathbf{P}}\mathbf{H}^{(\ell)} + \alpha^{(\ell)}\mathbf{H}^{(0)}\right). \tag{10}$$

And the final prediction result is

$$\widehat{\mathbf{Y}} = \mathrm{softmax}\left(\mathrm{MLP}\left(\mathbf{H}^{(L)}\right)\right). \tag{11}$$

3.5 The Data Augmentation

The problem of over-fitting will occur when DRGCN deals with a small dataset. To make DRGCN suitable for modeling data of various scales. We propose DRGCN* by referring to data argumentation and consistency regularization loss from GRAND [9]. Firstly, performing S times of data augmentation, naturally, we get the prediction result of the s-th data augmentation as $\widehat{\mathbf{Y}}^{(s)}(s = 1, \ldots, S)$ referring to Eq. 11. Suppose that the true label is $\mathbf{Y}$, and the supervised loss of the node classification task is defined as

$$\mathcal{L}_{\mathrm{sup}} = -\frac{1}{S}\sum_{s=1}^{S}\sum_{i=1}^{n}\mathbf{Y}_i\widehat{\mathbf{Y}}_i^{(s)}. \tag{12}$$

Secondly, we implement the consistency regularization loss to control the S prediction results to be as same as possible. We calculate the distribution center of the prediction label by $\overline{\mathbf{Y}}_i = \frac{1}{S}\sum_{i=1}^{s}\widehat{\mathbf{Y}}_i^{(s)}$. Then we utilize the sharpening trick [9] to get the labels based on the average distributions $\overline{\mathbf{Y}}_i \rightarrow \overline{\mathbf{Y}}_i'$. The consistency regularization loss $\mathcal{L}_{\mathrm{con}}$ is obtained by minimizing the distance between $\overline{\mathbf{Y}}_i'$ and $\mathbf{Y}_i^{(s)}$

$$\mathcal{L}_{\mathrm{con}} = \frac{1}{S}\sum_{s=1}^{s}\sum_{i=1}^{n}\left\|\overline{\mathbf{Y}}_i' - \widehat{\mathbf{Y}}_i^{(s)}\right\|_2^2. \tag{13}$$

Finally, when using λ to balance the two losses (as an usual case $\lambda = 1$), the final loss of DRGCN* is defined as

$$\mathcal{L} = \mathcal{L}_{\mathrm{sup}} + \lambda\mathcal{L}_{\mathrm{con}}, \tag{14}$$

By introducing these two training techniques, the over-fitting issue of DRGCN on small-scale dataset can be effectively solved. In addition, the computational complexity of DRGCN* is linear with the sum of node and edge counts [9].

4 EXPERIMENTS

In this section, we conduct extensive experiments on node classification tasks to evaluate the superiority of our proposed DRGCN. First, we introduce the datasets and experimental settings we used. Then we compare DRGCN with previous state-of-the-art deep graph neural network models to demonstrate the effectiveness of DRGCN on a variety of open graph datasets. To demonstrate the robustness of the model, experiments are carried out from two perspectives: model depths and training set sizes. Next, we present the ablation analysis of the model DRGCN for further verification. Then, we do extensive visualization analysis to demonstrate the potentially good interpretability of the proposed model. Finally, we do experiments on large-scale datasets.

Table 1: Summary of the datasets used in our experiments.

Dataset	Classes	Nodes	Edges	Features
Cora	7	2,708	5,429	1433
Citeseer	6	3,327	4,732	3703
Pubmed	3	1,9717	44,338	500
ogbn-arxiv	40	169,343	1,166,243	128
ogbn-products	47	2,449,029	61,859,140	100

4.1 Dataset and Experimental setup

First, we use three standard citation network datasets Cora, Citeseer, and Pubmed [31] for semi-supervised node classification. In these citation datasets, nodes correspond to documents, and edges correspond to citations; each node feature corresponds to the bag-of-words representation of the document and belongs to one of the academic topics. Then, to verify model performance for large-scale datasets, we also conduct the experiments on the Node Property Prediction of Open Graph Benchmark (OGBN)[16], which includes several various challenging and large-scale datasets. The statistics of overall datasets are summarized in Table 1.

In the experiments, we apply the standard fixed validation and testing split [18] on three citation datasets with 500 nodes for validation and 1,000 nodes for testing. In addition, in training set sizes experiments, we conduct experiments with different training set sizes {140,500,750,1000,1250} on the Cora dataset for several representative baselines. Then, according to the performance of the training set sizes experiments, we adopted fixed training set sizes for model depths experiments, which are 1000 for Cora, 1600 for Citeseer, and 10000 for Pubmed, and we aim to provide a rigorous and fair comparison between different models on each dataset by using the same dataset splits and training procedure. In addition, OGB dataset splitting directly calls the officially provided interface.

It should be noted that the effect of baseline models is greatly affected by model depths and training set sizes, therefore, we tune hyerparameters for all baseline models individually through grid search and some baselines even achieve better results than their original reports. Furthermore, the results of all the experiments are averaged over 20 runs with random weight initializations.

4.2 Baseline Models

We compare our DRGCN with the following baselines:

- **GCN** [18]: a graph convolutional neural network which uses an efficient layer-wise propagation rule that is based on a first-order approximation of spectral.
- **GAT** [37]: a novel convolution-style neural network that leverages masked self-attentional layers to address the shortcomings of prior methods based on graph convolutions or their approximations.
- **DAGNN** [26]: a deep adaptive GNN that focuses on incorporating information from large receptive fields through the entanglement of representation transformation and propagation. Specifically, DAGNN can leverage large receptive fields by decoupling this entanglement without suffering from performance deterioration.

- **GRAND** [9]: a simple yet effective framework that uses the random propagation strategy to stochastically generate multiple graph data augmentations, based on which it utilizes consistency regularization to improve the model ' s generalization on unlabeled data.
- **GCNII** [4]: a simple and deep GCN model that prevents over-smoothing by initial residual connection and identity mapping.
- **RevGAT** [22]: a deep gnn model which integrates reversible connections, group convolutions, weight tying, and equilibrium models to advance the memory and parameter efficiency of GNNs.

Table 2: Summary of classification accuracy(%) results with various depths. For a more rigorous comparison, all non "-Ori" baselines are well-tuned manually.

Dataset	Method	Layers					
		2	4	8	16	32	64
Cora	GCN	81.9	78.3	68.5	60.2	23.0	14.1
	GAT	83.5	79.3	65.2	60.1	35.3	16.2
	DAGNN	68.2	83.0	87.2	87.3	86.6	84.6
	GRAND	86.1	87.0	87.1	86.7	84.6	77.0
	GCNII	**86.6**	86.4	86.7	86.8	86.9	87.1
	GRAND-Ori	75.9	80.8	80.9	76.6	71.2	62.8
	GCNII-Ori	86.4	85.8	84.9	83.6	83.4	82.2
	DRGCN	85.9	86.7	87.0	87.3	87.3	87.5
	DRGCN*	**86.6**	**87.6**	**87.7**	**87.8**	**87.9**	**87.9**
Citeseer	GCN	76.8	66.2	32.4	14.4	14.2	10.2
	GAT	76.9	65.3	35.5	19.0	19.0	15.3
	DAGNN	72.6	73.8	76.2	76.4	76.2	76.0
	GRAND	77.4	78.2	78.0	76.9	75.8	75.0
	GCNII	77.6	77.6	78.2	78.1	78.2	78.3
	GRAND-Ori	77.4	78.0	77.7	76.9	75.7	64.2
	GCNII-Ori	77.5	77.3	77.2	77.0	76.3	75.7
	DRGCN	78.1	78.4	78.5	78.5	78.5	78.7
	DRGCN*	**78.4**	**78.6**	**78.8**	**78.6**	**78.8**	**78.9**
Pubmed	GCN	87.6	85.6	65.3	45.1	18.0	17.9
	GAT	87.9	86.1	66.3	51.1	23.2	23.2
	DAGNN	84.8	85.9	86.0	84.9	84.0	82.8
	GRAND	88.3	88.2	87.4	86.5	85.3	83.6
	GCNII	**88.9**	88.1	88.9	89.0	89.2	89.3
	GRAND-Ori	88.3	88.2	87.2	86.2	84.6	75.3
	GCNII-Ori	88.7	87.0	86.1	86.9	86.8	86.7
	DRGCN	88.6	89.0	89.1	89.5	89.3	89.6
	DRGCN*	88.8	**89.2**	**89.8**	**89.7**	**89.8**	**89.9**

4.3 Model Depths

This section will show the performance of the DRGCN compared to other state-of-the-art models as the graph neural network becomes deeper. We use the Adam SGD optimizer [17] with a learning rate of 0.001 and early stopping with the patience of 500 epochs to train DRGCN and DRGCN*. We set L2 regularization of the convolutional layer, fully connected layer, and evolving layer to 0.01, 0.0005, and 0.005 respectively. For baseline models, as the model layers increases, the hyperparameters which are provided by the origin paper are no longer applicable. We tune hyperparameters for baseline models individually through grid search.

Table 3: Summary of best classification accuracy(%) results of each model on Cora, Citeseer, and Pubmed. The number in parentheses corresponds to the depth of each model.

Model	Cora	Citeseer	Pubmed
GCN	81.9 ± 0.5 (02)	76.8 ± 0.5 (02)	87.6 ± 0.5 (02)
GAT	83.2 ± 0.5 (02)	76.9 ± 0.5 (02)	87.9 ± 0.5 (02)
DAGNN	87.3 ± 0.1 (16)	76.4 ± 0.1 (16)	86.0 ± 0.1 (08)
GRAND	87.1 ± 0.1 (08)	78.2 ± 0.1 (04)	88.3 ± 0.1 (02)
GCNII	87.1 ± 0.3 (64)	78.3 ± 0.3 (64)	89.3 ± 0.3 (64)
DRGCN	87.5 ± 0.2 (64)	78.7 ± 0.2 (64)	89.6 ± 0.2 (64)
DRGCN*	**87.9** ± 0.1 (64)	**78.9** ± 0.1 (64)	**89.9** ± 0.1 (64)

Table 4: Summary of classification accuracy(%) results with different training set sizes on Cora. For a more rigorous comparison, all non "-Ori" baselines are well-tuned manually.

Model	Training Set Sizes				
	140	500	750	1000	1250
GCN	80.3	81.3	81.7	81.9	82.4
GAT	83.1	83.2	82.7	83.5	83.7
DAGNN	84.4	86.8	87.0	87.3	87.5
GRAND	85.5	86.3	86.9	87.1	87.3
GCNII	85.3	85.9	86.5	87.1	87.0
GRAND-Ori	85.5	83.1	82.3	80.7	79.7
GCNII-Ori	85.3	84.4	83.5	82.7	81.5
DRGCN	58.0	73.9	80.3	87.5	87.5
DRGCN*	**85.8**	**86.4**	**87.2**	**87.9**	**87.7**

4.3.1 A Detailed Comparison with Other Models. Table 2 summaries the results for the deep models with various numbers of layers. We observe that on Cora, Citeseer, and Pubmed, DRGCN and DRGCN* achieve state-of-the-art performance, moreover, the performance of DRGCN and DRGCN* consistently improves as we increase the model depth. Overall, the performance of GCN and GAT drops rapidly with the increase of layers, DAGNN and GRAND show a trend of first rising and then falling as the number of GCN layers increases, which means they still suffer from over-smoothing. In addition, GRAND-Ori and GCNII-Ori represent the performance of using the best hyperparameters provided by the original paper. Here it can be concluded that GCNII-Ori and GRAND-Ori perform badly without tuning hyperparameters, while our proposed DRGCN and DRGCN* avoid time-consuming hyperparameter tuning. The best hyperparameters of the baseline models will be given in detail in the appendix.

4.3.2 Comparison with SOTA. Table 3 is a condensed version of Table 2, which summarizes the optimal results of each model on three citation datasets. As can be seen from Table 3, experimental results successfully demonstrate that DRGCN and DRGCN* achieve new state-of-the-art performance compared with baseline models. In addition, DRGCN* is the most stable model according to experimental variances.

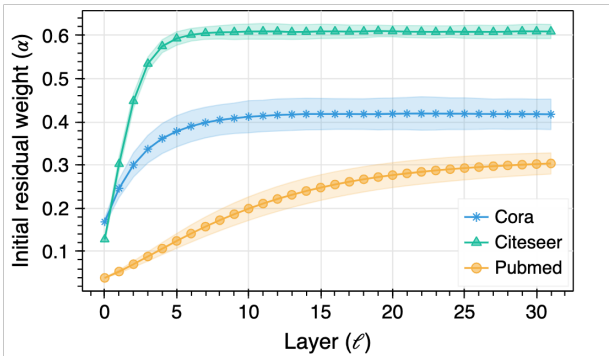

Figure 3: The evolving curve of dynamic residual from shallow to deep layers learned by DRGCN on Cora, Citeseer, and Pubmed. Each marker is the average α of all the nodes in the corresponding layer. The shaded areas represent the 95% confidence interval of multiple-time experiments.

4.4 Training Set Sizes

The previous section significantly demonstrates the effectiveness of our proposed model when the graph becomes deeper. In this section, we conduct experiments with different training set sizes to further demonstrate that DRGCN and DRGCN* are capable of capturing information from different scales of datasets. As shown in Table 4, to maximize the performance of baselines on different training data scales, we tune hyperparameters separately for each baseline model. GRAND and GCNII have a set of hyperparameters for each training size, GRAND-Ori and GCNII-Ori use the best hyperparameters provided by the original paper. See appendix for more hyperparameters information.

The following conclusions can be drawn from Table 4. Firstly, our DRGCN and DRGCN* have the best performance compared to other baseline models on training data with different scales. Secondly, the performance of the *Ori* version indicates that untuned GRAND and GCNII are not robust enough for different training set sizes. Thirdly, because DRGCN and DRGCN* can adaptively learn dynamic initial residual via evolving, they performed stable for different training set sizes without tuning hyperparameters.

4.5 Ablation Study

Table 5 shows the results of an ablation study that evaluates the contributions of our three techniques: the dynamic initial residual module, the evolving initial residual module, and data augmentation with consistency regularization loss. We can make the following observations.

The results of Static-Ori indicate that original GCNII cannot adapt to different layers. Then, static-tuned has improved after well-tuned hyperparameters which indicates the importance of self-adaptive learning. Therefore, when using only the dynamic initial residual module, results are almost the same as static-tuned which shows "+Dyn" can learn adaptively. In addition, using only the evolving initial residual module can also get competitive results by learning dynamic residual dependencies between layers. Moreover, when combining dynamic initial residual with the evolving mechanism (i.e. DRGCN), the model has been significantly improved compared to "Static-Ori" with good adaptiveness. Finally, when we add data augmentation and consistency regularized loss to DRGCN (i.e. DRGCN*), we achieve further improvement.

Table 5: Ablation study of classification accuracy(%) results for DRGCN and DRGCN* on Cora. "Static-Ori" means original GCNII. "Static-Tuned" means well-tuned GCNII for each layer number. "+Dyn" means only using the dynamic initial residual module, and "+Evo" means only using the evolving module. "+Dyn & Evo" denotes DRGCN, and "+Dyn & Evo & Aug" denotes DRGCN*.

Model	Layers					
	2	4	8	16	32	64
Static-Ori	86.4	85.8	84.9	83.6	83.4	82.2
Static-Tuned	**86.6**	86.4	86.7	86.8	86.9	87.1
+Dyn	85.4	86.2	86.4	86.5	86.6	86.9
+Evo	85.6	86.4	86.7	87.1	87.2	87.2
+Dyn & Evo	85.9	86.7	87.0	87.3	87.3	87.5
+Dyn & Evo & Aug	**86.6**	**87.6**	**87.7**	**87.8**	**87.9**	**87.9**

4.6 Visualization

In order to show that we do achieve dynamic residual for GCN, in this section, we visually study the characteristic of residual weights learned by DRGCN from two aspects: (1) evolving pattern analysis of dynamic residual; (2) distribution analysis of dynamic residual.

4.6.1 Evolving Pattern Analysis of Dynamic Residual. As shown in Figure 3, we visualize the evolving curve about dynamic residual learned by DRGCN on Cora, Citeseer, and Pubmed. Through analysis, we make three conclusions. Firstly, with the continuous iteration of the training epoch, the dynamic residual weight α in each layer converges to the curve shown in figure 3. It can be seen that α is small in shallow layers, and it gradually becomes larger and converges to a constant value as the model depth increases. The overall evolving trend of multiple datasets is consistent. Secondly, the convergence value of α on multiple datasets is different, which indicates that the evolving pattern of different datasets is also various. Thirdly, the inflection points of the curves are different, indicating that the boundary between short- and long-distance information is different for multiple datasets. In Figure 4, we show the detailed convergence process of the dynamic initial residual weight α on Cora, Citeseer, and Pubmed during the continuous training of the model DRGCN. Moreover, it converges eventually no matter what the initial value of α is.

4.6.2 Distribution Analysis of Dynamic Residual. Figure 5 shows the quartile chart of the dynamic initial residual weight α in each layer after DRGCN converges. We can summarize the following three observations. Firstly, DRGCN learns personalized dynamic initial residual weight α for each node in different layers. Secondly, the distribution trend of the dynamic residual α also has an evolving process, and the overall trend is consistent with Figure 3. Thirdly, in general, the distribution of α is more dense and consistent in shallow layers, and as the model deepens it becomes more dispersed, which suggests that the individual differences of long-distance information

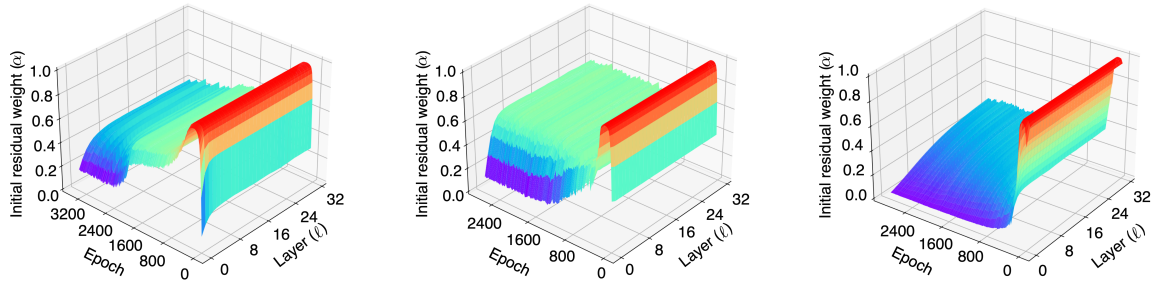

Figure 4: The convergence curve of the dynamic α learned by DRGCN on Cora (left), Citeseer (middle) and Pubmed (right).

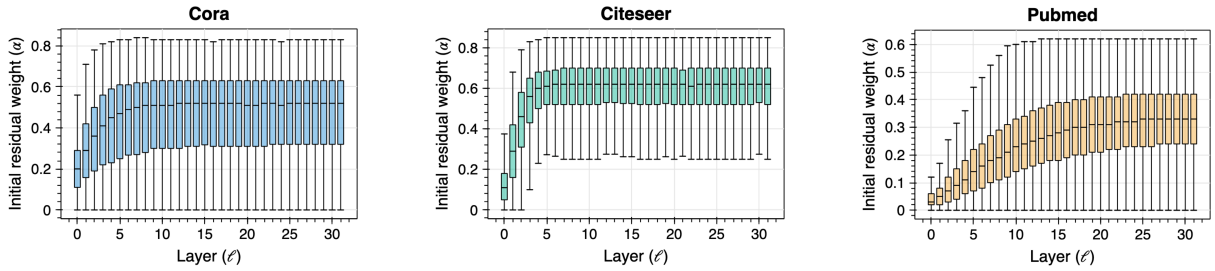

Figure 5: The quartile chart of the dynamic α learned by DRGCN in each layer on Cora, Citeseer, and Pubmed.

on the graph are more obvious and diverse, and it is more difficult to learn.

4.7 Large-Scale Datasets

Aligning to baseline models for solid comparison on small-scale datasets, the basic modules GCN and RNN are used in DRGCN. However, to improve the performance on large-scale datasets, GCN and RNN in DRGCN are upgraded to GAT and LSTM respectively, which is the DRGAT in Table 6. Then, we conduct experiments on ogbn-arxiv, and DRGAT achieves state-of-the-art performance compared to baseline models. Meanwhile, we propose the variant of DRGAT named DRGAT+GIANT-XRT+ SelfKD which uses self-knowledge distillation[47] and new features pre-trained by GIANT-XRT[6], and it reaches SOTA result on the ogbn-arxiv dataset of the OGB leaderboard[16] at the time of submission (team anonymous).

4.7.1 Full-Batch vs. Mini-Batch Training. In the previous section, DRGCN or DRGAT runs all experiments with full-batch training. In order to adapt our model to larger-scale scenarios, we propose a memory-efficient version DRGAT-Mini-Batch (Appendix contains complete Full-Batch and Mini-Batch pseudocode). First, we conduct several experiments to compare the performance of DRGAT-Full-Batch and DRGAT-Mini-Batch on Cora, ogbn-arxiv and ogbn-products. As shown in table 7 in the appendix, compared to full-batch training, we find that mini-batch training further greatly reduces the memory consumption of DRGAT without degrading model performance. In addition, the mini-batch version is affected by sampling, and the random sample is currently used. The optimization of sampling will further improve the performance of the model.

Table 6: Summary of classification accuracy(%) results on the ogbn-arxiv dataset compared to SOTA.

Model	Test Acc(%)
GCN	71.74 ± 0.29
DAGNN	72.09 ± 0.25
GCNII	72.74 ± 0.16
GAT	73.91 ± 0.12
RevGAT	74.06 ± 0.18
DRGAT	**74.16** ± 0.07
RevGAT+GIANT-XRT	75.90 ± 0.19
DRGAT+GIANT-XRT	**76.11** ± 0.09
RevGAT+GIANT-XRT+SelfKD	76.15 ± 0.10
DRGAT+GIANT-XRT+SelfKD	**76.33** ± 0.08

5 CONCLUSIONS AND FUTURE WORK

In this paper, we propose a novel deep residual GCN model named DRGCN, which can self-adaptively learn the residual weights for different layers. Comprehensive experiments show that our model relieves the over-smoothing issue and outperforms other state-of-the-art methods on both small-scale and large-scale benchmark datasets at the same time. Visualizing the convergence and distribution of residual weights in different layers suggests that our model can model the evolving process of residual weights, which consequently have a positive effect on final model performance

In future work, we will demonstrate that our model can be a general technique for all the GCN variants incorporating the residual connection on both homogeneous and heterogeneous graphs. Meanwhile, we will try our method with different graph tasks to further verify our superior performance.

REFERENCES

[1] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. 2019. MixHop: Higher-Order Graph Convolutional Architectures via Sparsified Neighborhood Mixing. In *Proceedings of the International Conference on Machine Learning (ICML*. 21–29.

[2] Rianne van den Berg, Thomas N Kipf, and Max Welling. 2017. Graph Convolutional Matrix Completion. *arXiv preprint arXiv:1706.02263* (2017).

[3] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. *International Conference on Learning Representations (ICLR)* (2018).

[4] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. 2020. Simple and Deep Graph Convolutional Networks. In *Proceedings of the International Conference on Machine Learning (ICML)*. 1725–1735.

[5] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.

[6] Eli Chien, Wei-Cheng Chang, Cho-Jui Hsieh, Hsiang-Fu Yu, Jiong Zhang, Olgica Milenkovic, and Inderjit S Dhillon. 2021. Node Feature Extraction by Self-Supervised Multi-scale Neighborhood Prediction. *arXiv preprint arXiv:2111.00064* (2021).

[7] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. *Advances in Neural Information Processing Systems (NeurIPS)* 29 (2016), 3844–3852.

[8] Zhijie Deng, Yinpeng Dong, and Jun Zhu. 2019. Batch Virtual Adversarial Training for Graph Convolutional Networks. *arXiv preprint arXiv:1902.09192* (2019).

[9] Wenzheng Feng, Jie Zhang, Yuxiao Dong, Yu Han, Huanbo Luan, Qian Xu, Qiang Yang, Evgeny Kharlamov, and Jie Tang. 2020. Graph Random Neural Network for Semi-Supervised Learning on Graphs. *Advances in Neural Information Processing Systems (NeurIPS)* (2020).

[10] Shengnan Guo, Youfang Lin, Ning Feng, Chao Song, and Huaiyu Wan. 2019. Attention Based Spatial-Temporal Graph Convolutional Networks for Traffic Flow Forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 922–929.

[11] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs.

[12] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation Learning on Graphs: Methods and Applications. *arXiv preprint arXiv:1709.05584* (2017).

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778.

[14] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*. 639–648.

[15] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. 1989. Multilayer Feedforward Networks Are Universal Approximators. *Neural networks* 2, 5 (1989), 359–366.

[16] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *arXiv preprint arXiv:2005.00687* (2020).

[17] Diederik P Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980* (2014).

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

[19] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. *International Conference on Learning Representations (ICLR)* (2019).

[20] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. 2019. Diffusion Improves Graph Learning. *Advances in Neural Information Processing Systems (NeurIPS)* 32 (2019), 13354–13366.

[21] Guohao Li, Matthias Müller, Ali K. Thabet, and Bernard Ghanem. 2019. DeepGCNs: Can GCNs Go As Deep As CNNs?. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 9267–9276.

[22] Guohao Li, Matthias M ü ller, Bernard Ghanem, and Vladlen Koltun. 2021. Training Graph Neural Networks with 1000 layers. In *International Conference on Machine Learning (ICML)*.

[23] Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. 2020. DeeperGCN: All You Need to Train Deeper GCNS. *arXiv preprint arXiv:2006.07739* (2020).

[24] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper Insights Into Graph Convolutional Networks for Semi-Supervised Learning. In *Proceddings of the Thirty-Second AAAI conference on Artificial Intelligence*.

[25] Ruoyu Li, Sheng Wang, Feiyun Zhu, and Junzhou Huang. 2018. Adaptive Graph Convolutional Neural Networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

[26] Meng Liu, Hongyang Gao, and Shuiwang Ji. 2020. Towards Deeper Graph Neural Networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 338–348.

[27] Jiaqi Ma, Weijing Tang, Ji Zhu, and Qiaozhu Mei. 2019. A Flexible Generative Framework for Graph-based Semi-supervised Learning. *Advances in Neural Information Processing Systems (NeurIPS)* 32 (2019), 3281–3290.

[28] Federico Monti, Michael M Bronstein, and Xavier Bresson. 2017. Geometric Matrix Completion with Recurrent Multi-Graph Neural Networks. *Advances in Neural Information Processing Systems (NeurIPS)* (2017).

[29] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank Citation Ranking: Bringing Order to The Web.* Technical Report. Stanford InfoLab.

[30] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. 2020. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. *International Conference on Learning Representations (ICLR)* (2020).

[31] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective Classification in Network Data. *AI magazine* 29, 3 (2008), 93–93.

[32] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. 2021. Masked Label Prediction: Unified Message Passing Model for Semi-Supervised lassification. *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI)* (2021).

[33] Ke Sun, Zhanxing Zhu, and Zhouchen Lin. 2021. AdaGCN: Adaboosting Graph Convolutional Networks into Deep Models. *International Conference on Learning Representations (ICLR)* (2021).

[34] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. 2017. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

[35] Kiran K Thekumparampil, Chong Wang, Sewoong Oh, and Li-Jia Li. 2018. Attention-based Graph Neural Network for Semi-supervised Learning. *arXiv preprint arXiv:1803.03735* (2018).

[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All You Need. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[37] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. *International Conference on Learning Representations (ICLR)* (2018).

[38] Vikas Verma, Meng Qu, Alex Lamb, Yoshua Bengio, Juho Kannala, and Jian Tang. 2019. GraphMix: Regularized Training of Graph Neural Networks for Semi-Supervised Learning. (2019).

[39] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. 2019. Heterogeneous Graph Attention Network. In *The World Wide Web Conference (WWW)*. 2022–2032.

[40] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. 2018. SkipNet: Learning Dynamic Routing in Convolutional Networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 409–424.

[41] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying Graph Convolutional Networks. In *International Conference on Machine Learning (ICML)*. 6861–6871.

[42] Bingbing Xu, Huawei Shen, Qi Cao, Yunqi Qiu, and Xueqi Cheng. 2019. Graph Wavelet Neural Network. *arXiv preprint arXiv:1904.07785* (2019).

[43] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation learning on graphs with jumping knowledge networks. In *Proceedings of the International Conference on Machine Learning (ICML)*. 5453–5462.

[44] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 974–983.

[45] Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2018. Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting. *International Joint Conference on Artificial Intelligence (IJCAI)* (2018).

[46] Jiani Zhang, Xingjian Shi, Junyuan Xie, Hao Ma, Irwin King, and Dit-Yan Yeung. 2018. GaAN: Gated Attention Networks for Learning on Large and Spatiotemporal Graphs. *Conference on Uncertainty in Artificial Intelligence (UAI)* (2018).

[47] Linfeng Zhang, Jiebo Song, Anni Gao, Jingwei Chen, Chenglong Bao, and Kaisheng Ma. 2019. Be your own teacher: Improve the performance of convolutional neural networks via self distillation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 3713–3722.

[48] Lingxiao Zhao and Leman Akoglu. 2019. PairNorm: Tackling Oversmoothing in GNNs. *arXiv preprint arXiv:1909.12223* (2019).

[49] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2018. Graph Neural Networks: A Review of Methods and Applications. *AI Open* (2018).

[50] Jun Zhu, Jiaming Song, and Bei Chen. 2016. Max-Margin Nonparametric Latent Feature Models for Link Prediction. *arXiv preprint arXiv:1602.07428* (2016).

A APPENDIX

A.1 Pseudocode

Pseudocode of Full-Batch and Mini-Batch versions of DRGAT are shown in Algorithm 1 and 2, respectively.

Algorithm 1 DRGAT-Full-Batch

Input: $\mathbf{A}$: adjacency matrix, $\mathbf{X}$: node features, L: gnn depth

Output: predictions of model

$\mathbf{H}^{(0)} \leftarrow \text{MLP}(\mathbf{X})$
$\widetilde{\mathbf{H}}^{(0)} \leftarrow \mathbf{H}^{(0)} / \|\mathbf{H}^{(0)}\|_2$
for $\ell = 0$ to $L-1$ **do**
  $\widetilde{\mathbf{H}}^{(\ell)} \leftarrow \mathbf{H}^{(\ell)} / \|\mathbf{H}^{(\ell)}\|_2$
  $\mathbf{z}^{(\ell)} \leftarrow \text{MLP}\left(\Phi\left(\widetilde{\mathbf{H}}^{(0)}, \widetilde{\mathbf{H}}^{(\ell)}\right)\right)$
  $\boldsymbol{\alpha}^{(\ell)} \leftarrow \text{RNN}\left(\mathbf{z}^{(\ell)}\right)$
  $\mathbf{A}_{att} \leftarrow \text{GAT}\left(\mathbf{A}, \mathbf{H}^{(\ell)}\right)$
  $\mathbf{H}^{(\ell+1)} \leftarrow \mathbf{A}_{att}\mathbf{H}^{(\ell)}$
  $\mathbf{H}^{(\ell+1)} \leftarrow \text{ReLU}\left(\left(1-\boldsymbol{\alpha}^{(\ell)}\right)\mathbf{H}^{(\ell+1)} + \boldsymbol{\alpha}^{(\ell)}\mathbf{H}^{(0)}\right)$
end for
$out \leftarrow \text{Softmax}\left(\text{MLP}\left(\mathbf{H}^{(L)}\right)\right)$

A.2 Hyperparameters Details

In this section, partial important hyperparameters of some models are presented in Table 8 and 9.

A.3 Experimental Details

Table 10 and 11 give more detailed experimental data for Figure 1.

Algorithm 2 DRGAT-Mini-Batch

Input: $\mathcal{G}(\mathcal{V}, \mathcal{E})$: graph, $\mathbf{x}_v$: node features for v, $\mathcal{B}$: current batch, K: #hop, L: gnn depth, $\mathcal{N}_k$: neighborhood sampling function for k-th hop

Output: predictions of model

$\mathcal{B}^{(K)} \leftarrow \mathcal{B}$
for $k = K$ to 1 **do**
  $\mathcal{B}^{(k-1)} \leftarrow \mathcal{B}^{(k)}$
  for $u \in \mathcal{B}^{(k)}$ **do**
    $\mathcal{B}^{(k-1)} \leftarrow \mathcal{B}^{(k-1)} \cup \mathcal{N}_k(u)$
  end for
end for
$\mathbf{h}_v^{(0)} \leftarrow \text{MLP}(\mathbf{x}_v), \forall v \in \mathcal{B}^{(0)}$
for $\ell = 0$ to $L-1$ **do**
  for $v \in \mathcal{B}^{(0)}$ **do**
    $\mathbf{h}_v^{(\ell)} \leftarrow \mathbf{h}_v^{(\ell)} / \|\mathbf{h}_v^{(\ell)}\|_2$
    $\mathbf{h}_{\mathcal{N}(v)}^{(\ell+1)} \leftarrow \text{AGGREGATE}_{\ell+1}\left(\left\{\mathbf{h}_{v'}^{(\ell)}, \forall v' \in \mathcal{N}(v)\right\}\right)$
    $\mathbf{h}_v^{(\ell+1)} \leftarrow \sigma\left(\mathbf{W}^{(\ell+1)} \cdot \text{CONCAT}\left(\mathbf{h}_v^{(\ell)}, \mathbf{h}_{\mathcal{N}(v)}^{(\ell+1)}\right)\right)$
    $\mathbf{z}_v^{(\ell+1)} \leftarrow \text{MLP}\left(\Phi\left(\mathbf{h}_v^{(0)}, \mathbf{h}_v^{(\ell+1)}\right)\right)$
    $\alpha_v^{(\ell+1)} \leftarrow \text{RNN}\left(\mathbf{z}_v^{(\ell+1)}\right)$
    $\mathbf{h}_v^{(\ell+1)} \leftarrow \text{ReLU}\left(\left(1-\alpha_v^{(\ell+1)}\right)\mathbf{h}_v^{(\ell+1)} + \alpha_v^{(\ell+1)}\mathbf{h}_v^{(0)}\right)$
  end for
end for
$out \leftarrow \text{Softmax}\left(\text{MLP}\left(\mathbf{h}_v^{(L)}\right)\right), \forall v \in \mathcal{B}$

Table 7: DRGAT-Full-Batch vs DRGAT-Mini-Batch on the Cora, ogbn-arxiv and ogbn-products.

	Cora		**ogbn-arxiv**		**ogbn-products**	
	Acc(%)	**Mem**(GB)	**Acc**(%)	**Mem**(GB)	**Acc**(%)	**Mem**(GB)
DRGAT-Full-Batch	87.90	1.28	74.16	8.96	-	-
DRGAT-Mini-Batch	87.80	0.49	74.08	2.27	82.30	10.38

Table 8: Main hyper-parameters for DRGCN and DRGCN*.

Model	**Hyper-Parameters**								
	layers	hidden	dropout	learning_rate	L2_conv	L2_mlp	L2_rnn	aug_K	drop_node
DRGCN	64	64	0.6	1e-3	1e-2	5e-4	5e-3	-	-
DRGCN*	64	64	0.6	1e-3	1e-2	5e-4	5e-3	6	0.1

Table 9: Main hyper-parameters for GRAND and GCNII at different training set sizes.

Model	**Hyper-Parameters**	**Training Set Sizes**				
		140	500	750	1000	1250
GRAND	layers	6	6	6	8	8
	sample	4	2	2	1	1
	hidden	32	32	32	64	64
	drop_node	0.2	0.2	0.2	0.0	0.0
	drop_rate	0.5	0.5	0.5	0.5	0.5
	learning_rate	0.01	0.01	0.01	0.01	0.01
	lamda	1.0	1.0	1.0	1.0	1.0
GCNII	alpha	0.1	0.2	0.3	0.3	0.3
	layers	64	64	64	64	64
	hidden	64	64	64	64	64
	drop_rate	0.6	0.6	0.6	0.6	0.6
	learning_rate	0.01	0.01	0.01	0.01	0.01

Table 10: More detailed experiments of the static residual (GCNII) on Cora. Results in the table demonstrate that static residual is sensitive to model depths and training set sizes.

Model-TrainSetSize-LayerNum	**Acc on different α settings**										
	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
GCNII-1000-02	86.41	**86.61**	86.16	85.72	85.36	84.52	82.42	80.07	77.58	74.95	72.85
GCNII-1000-04	75.35	**86.38**	85.56	85.61	85.15	83.51	82.33	79.45	77.54	75.62	73.01
GCNII-1000-08	52.30	84.80	**86.71**	86.03	85.60	83.88	82.19	79.91	78.07	75.82	72.98
GCNII-1000-16	44.43	84.96	**86.78**	86.60	85.93	84.52	82.78	80.63	78.35	75.87	73.13
GCNII-1000-32	31.90	83.72	86.71	**86.87**	86.12	84.83	82.85	80.53	78.25	75.91	73.24
GCNII-1000-64	31.90	82.51	86.74	**87.14**	86.08	84.60	82.86	80.54	78.26	76.10	73.14
GCNII-0140-64	19.64	**85.26**	84.37	82.35	80.32	77.47	74.57	71.32	67.08	62.73	58.08
GCNII-0500-64	31.90	84.37	**85.92**	85.48	84.25	82.89	81.13	78.30	75.42	72.78	70.02
GCNII-0750-64	31.90	83.39	86.30	**86.51**	85.16	83.58	81.76	79.53	77.32	74.17	71.33
GCNII-1000-64	31.90	82.51	86.74	**87.14**	86.08	84.60	82.86	80.54	78.26	76.10	73.14
GCNII-1250-64	31.90	81.32	86.75	**87.04**	85.89	84.43	83.50	81.28	78.92	76.81	74.63

Table 11: More detailed experiments of the static residual (GCNII) on Cora, Citeseer, and Pumbed. Results in the table demonstrate that static residual is sensitive to different datasets.

Datasets	**Acc on different α settings**																	
	0.01	0.03	0.05	0.06	0.08	0.1	0.12	0.14	0.16	0.18	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Cora	57.7	78.8	82.0	84.6	85.0	85.2	**85.9**	85.1	84.3	83.8	84.6	82.3	80.8	77.7	75.0	71.6	68.1	62.1
Citeseer	18.0	57.0	68.7	72.0	72.5	73.1	73.0	72.9	73.6	**73.6**	73.4	71.5	69.0	67.1	64.3	62.6	59.3	58.3
Pubmed	58.3	76.8	79.0	79.1	79.5	**80.2**	79.9	80.0	79.4	79.5	79.9	79.4	78.7	76.9	76.1	75.0	74.4	73.5