RandCluster-GCN: Simple, Efficient and Powerful

Chenxin Xiong
KAUST ID: 168449
chenxin.xiong@kaust.edu.sa

1. Introduction

Graph convolutional networks (GCNs) [17] have been drawing lots of attention and interests due to its power to address many non-Euclidean data based applications with very promising results. GCNs can be widely used to do graph-based semi-supervised node classification [17], link prediction [2], point cloud segmentation [20], 3D mesh models generations from single RGB images [19] and so on. Considering the scales of real-world data are usually large, training GCNs on large-scale graphs, point cloud remains challenging due to the memory issue. Take training GCN on a given graph for example, GCN uses a graph convolution operation to obtain node embeddings (features) layer by layer. At each layer, the embedding of a node is obtained by aggregating the embeddings of its neighbors, followed by a non-linear transformation. Thus, all the parameters and outputs of GCN are necessary to be stored for doing back-propagation, which results in the memory issue and the failure of training GCN on GPUs further, especially when GCN goes deeper.

In order to address the issue mentioned above, many work has been done to develop a scalable GCN training algorithm. The methods proposed can be roughly divided into two categories: sampling-based and clustering-based [15]. For sampling-based methods [8, 3, 4], they all propose to sample only a few neighbors for every node in every GCN layer in variant sample criteria to implement mini-batch SGD by reducing the size of intermediate embeddings for every layer. However, all of these methods have non-negligible drawbacks from memory requirement, time per epoch or convergence speed (loss reduction) per epoch evaluations [5]. As a clustering-based method come up with firstly, Cluster-GCN [5] proposes to design the batches using an efficient graph clustering algorithm called METIS [16]. In order to address the issue of the skewed label distribution within each batch for some specific datasets, they use stochastic multiple clustering approach to reincorporate between-cluster links and make the variance across batches smaller through the combinations of clusters. For Reddit dataset, they partition the graph into 1500 (a relatively large number) clusters based on METIS algorithm, and then randomly select 20 clusters to form a batch during each training iteration.

In Cluster-GCN work, they claim that clustering partition outperforms random partition when keeps the number of clusters same because clustering partition has less between-partition links removed. They also point out that one of the advantages of doing random clustering is the variance across different batches will be extremely small, which is crucial to the the convergence of SGD. Due to insufficient experiments and analysis of the comparisons between random clustering and METIS clustering done in Cluster-GCN work, the conclusion drawn may be incorrect. Although random clustering causes more loss of edges, its main advantage is very crucial and largely influences GCN's performance.

In this work, we doubt the conclusion drawn based on insufficient experimental results and analysis in Cluster-GCN work that random clustering performs worse that METIS clustering. By doing extensive experiments and analysis, We argue that random clustering this simplest and most efficient clustering method is powerful enough as long as it is used in a correct and proper way (e.g., partitioning graph into a small number of clusters). We also would like to prove that not only is random clustering useful to training GCNs on large-scale graph, but also point cloud and 3D mesh, indicating random clustering could be a universe solution for large-scale GCNs training.

Based on the very high memory and computational efficiency brought by random clustering operation, training deeper GCNs is scalable and won't be restricted by the limited memory issues while the common vanishing gradient problem may still exist. Fortunately, motivated by the huge success of ResNet [10], DenseNet [13] used to train very deep CNNs, DeepGCNs [18] proposes to adapt these concepts to train very deeper GCNs (56 layers) successfully. Therefore, applying either residual connection or dense connections could prevent deep GCNs from suffering vanishing gradient problem and degradation issue, thus the ability of deeper GCNs could be taken advantage of to increase models' performances in a straightforward way.

2. Related Work

2.1. GCNs on large-scale graph

As mentioned previously, several sampling methods are proposed to address the issue of training GCNs on large-scale graph.

GraphSAGE [8] is a framework proposed to train GCNs by mini-batch SGD in an inductive way. The main idea of GraphSAGE is learning a function that generates embeddings by sampling and aggregating features from a node's local neighborhood. In this work, they uniformly sample a fixed-size set of neighbors for each node and draw different uniform samples at each iteration. In FastGCN [3] work, the major distinction from GraphSAGE is that this method samples vertices rather than neighbors and it is enhanced with importance sampling. VR-GCN [4] proposes to use a variance reduction technique to reduce the size of neighborhood sampling nodes.

Cluster-GCN [5] as a novel clustering-based GCN algorithm that is suitable for SGD-based training by exploiting the graph clustering structure. By using METIS clustering approach proposed, Cluster-GCN significantly improves memory and computational efficiency due to the restriction of the neighborhood search within each batch. Therefore, this method can train very deep GCNs on large-scale graph successfully, which makes Cluster-GCN achieve state-of-the-art test F1 score on PPI and Reddit datasets. For other datasets, Cluster-GCN is also able to achieve comparable test accuracy with previous sampling-based algorithms. Cluster-GCN will be mainly focused on as it's most relevant to our work.

2.2. GCNs on large-scale point cloud and 3D mesh

Semantic segmentation of large-scale 3D point clouds is still a quite challenging work,although many approaches proposed have achieved impressive results, however almost all of them are limited to extremely small 3D point clouds and cannot be directly extended to larger point clouds [12]. In this recent work [12], they introduce and compare the existing point sampling approaches and finally choose simple but highly efficient random sampling to decrease point density greatly. They also empirically prove the efficiency of random sampling.

For 3D mesh, Pixel2Mesh [19] proposes an deep learning algorithm to extract a 3D triangular mesh from a single color image, instead of generating a volume [6] or point cloud [7], which both lose important surface details. In this work, a graph based fully convolutional network (GCN) is directly built on the mesh model, where the vertices and edges in the mesh are represented as nodes and connections in a graph. MeshCNN: a neural network akin to the well-known CNN, but designed specifically for meshes. MeshCNN [9], is a neural network akin to the well-known CNN designed specifically for meshes. It operates directly on irregular triangular meshes, performing convolution and pooling operations designed in harmony with the unique mesh properties.

3. Cluster-GCN

Before propose our solution, we would like to analyse Cluster-GCN work and its performances on PPI and Reddit these two datasets.

As mentioned before, they observe that using clustering partition can achieve higher accuracy in comparison to random partition by doing experiment of partitioning the graph into 10 parts by using random partition and METIS respectively on Cora, Pubmed and PPI these three datasets. We notice that for PPI dataset, the random partition performs largely worse than clustering partition while that doesn't happen to other two datasets, as shown in Table 1, which attracts our interest. Moreover, except PPI, Reddit as another datasete Cluster-GCN achieves state-of-the-art on will be analysed as well.

Dataset	**random**	**clustering**	**difference**
Cora	78.4	82.5	-4.1
Pubmed	78.9	79.9	-1.0
PPI	68.1	92.9	**-24.8**

Table 1: Random partition versus clustering partition in terms of test F1 score.

The information of these two datasets is described statistically in Table 2.

Dataset	**Task**	**#Nodes**	**#Edges**	**#Labels**
PPI	multi-label	56,944	818,716	121
Reddit	multi-class	232,965	11,606,919	41

Table 2: Basic information of PPI and Reddit dataset respectively.

As mentioned in Cluster-GCN, clustering partitioned batches have low label entropy, indicating skewed label distribution within each batch. In other worlds, the variance across different batches is large, which is crucial to the convergence of SGD. Therefore, in the following content, we will compare random partition with clustering partition in terms of the distributions of labels (PPI) or classes (Reddit) across different batches by calculating the mean and standard variance of the differences between the distribution of training nodes and each batch.

Let P indicate the labels (classes) distribution of training nodes of a graph, while $Q_i (i = 1, 2, 3, ..., n)$ represents the distribution of ith batch after the graph is partitioned into n clusters by either random or clustering way. The similarity between two probability distributions P and Q_i is measured by JSD (Jensen–Shannon divergence) with the considerations of it's symmetry and it always has a finite value.

$$JSD(P||Q_i) = \frac{1}{2}D(P||M) + \frac{1}{2}D(Q_i||M) \qquad (1)$$

Where $M = \frac{1}{2}(P+Q_i)$ and D represents Kullback–Leibler divergence. Therefore, the mean (M) and standard variance (S) are formulated as follows:

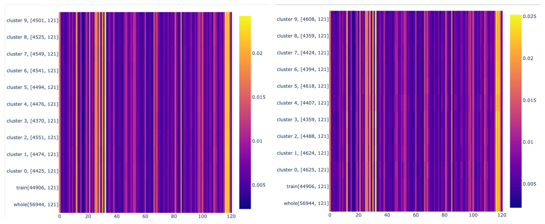
$$M = \frac{1}{n}\sum_{i=1}^{n} JSD(P||Q_i) \qquad (2)$$

$$S = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(JSD(P||Q_i) - M)^2} \qquad (3)$$

Through M and S, we are able to quantify the variance across distinct batches in a straightforward way.

3.1. PPI Dataset

Firstly, We visualize the labels' distributions of batches constructed by random partition 1a and clustering partition (Figure 1b) and calculate their corresponding M and S for PPI dataset when the number of clusters is set 10 on PPI. Because the variances by random and clustering are both extremely small, resulting in the main advantage of random partition is not able to be highlighted, thus clustering marginally outperforms random since it removes less between-partition links.



(a) Random partition (b) Clustering partition

Figure 1: There are 121 columns totally and each column indicates one label. The distinct colors filled in each row mean the different degree of proportion each label occupies. The first and second rows represent the labels' distribution over all nodes and training nodes respectively. Starting from the third row, every different batch is identified by each row. By observing the color consistency along each column from bottom to the top of the heatmap, we can roughly tell that the variance across all batches is very small either by random or clustering partition, which is testified by the values of M and S in Table 3.

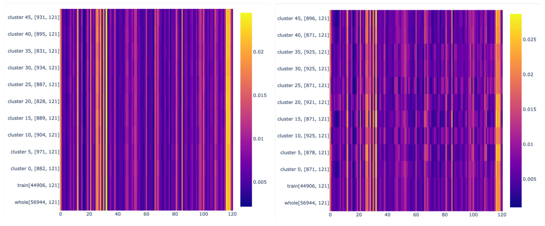
#clusters=10	M	S
Random	0.0078	0.0
Clustering	0.0147	0.0

Table 3: M and S calculated by Equation 2, 3

Then we set the number of clusters to 50, which is the exact number chosen in Cluster-GCN for obtaining the final state-of-the-art result. We also rerun the code and get the best test F1 score by doing random and clustering partition respectively. The results are shown in Table 5. Now, because random partition keeps the distributions of batches almost be the same with that of training nodes, which largely makes up the degradation on the performance caused by more edges lost, that's the reason why two F1 scores don't have too much difference.

#clusters=50	M	S	**Test F1 Score**
Random	0.0187	0.0	0.97304
Clustering	0.0902	0.02	0.99319

Table 4: M, S and the best test F1 score.



(a) Random partition (b) Clustering partition

Figure 2: When the number of clusters is 50, it can be apparently observed that the consistency of colors along each column in 2b is not as good as that of in 2a.

3.2. Reddit Dataset

For solving the unbalanced classes distribution by using the Reddit data with clusters formed by METIS, The method applied to Reddit is different from that applied to PPI by a stochastic multiple clustering approach to incorporate between-cluster links and reduce variance across batches.

When the number of clusters is 300 and the batch size is 1 (no between-cluster links incorporate), we can apparently notice that the nodes with the same class are tend to be brought together by METIS algorithm (see Figure 4), which leads to a biased estimation of the full gradient while performing SGD updates. Moreover, the test results obtained are consistent with PPI as Table .

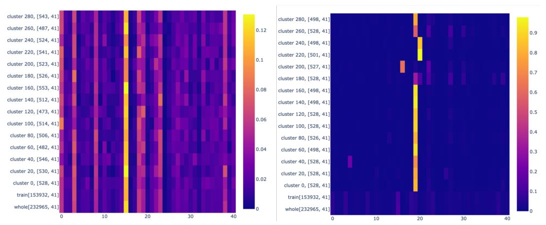

(a) Random partition (b) Clustering partition

Figure 3: For clustering partition, most of batches have very different classes distributions from training nodes. Each batch is mainly formed by similar nodes.

#clusters=300	M	S	Test Accuracy
Random	0.1221	0.014	0.95814
Clustering	0.8333	0.0877	0.96172

Table 5: M, S and the best test accuracy.

In order to address this issue, Cluster-GCN proposes a stochastic multiple clustering approach. For Reddit, they firstly partition the graph into a relatively large number N of clusters using METIS algorithm and then form each batch by randomly selecting a small number BS of (batch size) clusters, thus there are $(\frac{N}{BS})$ batches during each epoch. More details about the algorithm can be checked in [5].

Now It is able to be known that the reason why Reddit dataset should be partitioned into a relatively large number of clusters and then form a batch using a small number of clusters selected randomly is that more randomness will be incorporated by this way for balancing the biased estimation of the full gradient caused by the large variance across the batches, while PPI doesn't need due to it has almost the same small enough M and S values with random which brings little influence on the performance even if the number of clusters is very small (10).

#clusters=300	batch size=1	Test Accuracy
Random		0.95814
Clustering		0.96172
#clusters=1500	**batch size=20**	**Test Accuracy**
Random		0.95825
Clustering		0.96172

Table 6: Best test accuracy.

In Table 6 we compare the test accuracy and we find out that for both 300(1) and 1500(20), random partition just slightly worse than clustering partition. 300(1) happens to have the same accuracy with 1500(20) for clustering partition, which may because the former keeps more edges and the latter's variance of batches is much smaller. However this kind of comparison is unfair cause the total times of iterations are different.

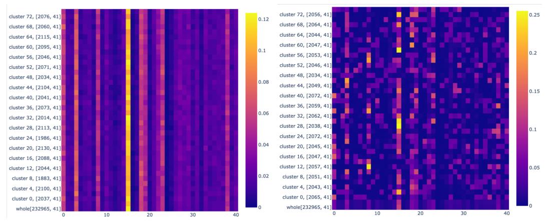

(a) Random partition (b) Clustering partition

Figure 4: The whole graph is partitioned into 1500 clusters by random and clustering respectively and each batch is formed by 20 clusters selected randomly, which results in 75 batches totally (for random, actually the graph can be just partitioned into 75 clusters directly, but we apply the same operation with clustering partition just for convenience). By this stochastic way, variance cross the batches can be decreased obviously. The comparison is shown in Table 7.

#clusters	batch size	M	S
300	1	0.8333	0.0877
1500	20	0.0902	0.02

Table 7: M, S comparison before and after use the stochastic multiple clustering approach.

Based on the experimental results shown above, relevant analysis and explanations. We argue that random partition may be a more simple, efficient solution used to be designed a scalable GCN algorithm than Cluster-GCN for some reasons as follows:

- Random partition reduces the manual design about the number of clusters partitioned and batch size that apparently vary for different datasets.
- Random partition has highest efficiency and most straightforward implementation, no other complicated algorithms involved.
- Even we haven't made any extra efforts to make up its main drawback that more edges are lost, its performance is comparable with clustering partition already, indicating its potential to outperform clustering partition.

In the following, we would like to use random clustering instead of random partition to describe the operation of partitioning the graph randomly.

4. Random Clustering

In order to improve random clustering, the main drawback that the large loss of edges' information as mentioned before should be considered and solved firstly. We propose RandCluster-GCN based on some criteria of doing random clustering properly as follows:

Criterion 1. Extra nodes features from the information provided by edges before partition the graph. For some datasets, there are only node features provided, while for same, the edges have their own features as well. According to these two different types of datasets, the nodes features could be constructed by aggregating either the features of edges each node connected directly or the features of neighbors of each node.

Criterion 2. Do random clustering to generate batches each epoch rather than doing random clustering just once. In the following, we will use **multi-random** to describe the former operation and use **single-random** for the later. The reason is apparent and straightforward because the edges lost by **single-random** won't be retrieved and they are impossible to be utilized during training phase forever. In contract, **multi-random** is able to cover all edges when the number of training epochs is big enough. During each epoch, the edges haven't been used yet have probabilities to be selected. We would like to prove this claim empirically in the Experiment section.

Criterion 3. Partition the graph into a small enough number of clusters (this number could be determined by the scale of data and GPU memory). This operation is trying to keep as many edges as possible, especially for the inference phase. For training process, there exists a trade-off between the number of edges kept and the speed of convergence (partition the graph into too small number of clusters may result in slow convergence and worse performance even less edges lost at the same times of iterations).

5. Experiments

5.1. Dataset

The dataset we use is ogbn-proteins, its information is shown in Table 8.

Task	#Nodes	#Edges	#Labels
multi-label	132,534	39,561,252	112

Table 8: ogbn-proteins is an undirected, weighted, and typed (according to species) graph. Nodes represent proteins, and edges indicate different types of biologically meaningful associations between proteins. All edges come with 8-dimensional features and each node comes with an 8-dimensional one-hot feature indicating which species the corresponding protein comes from (see Figure 5).

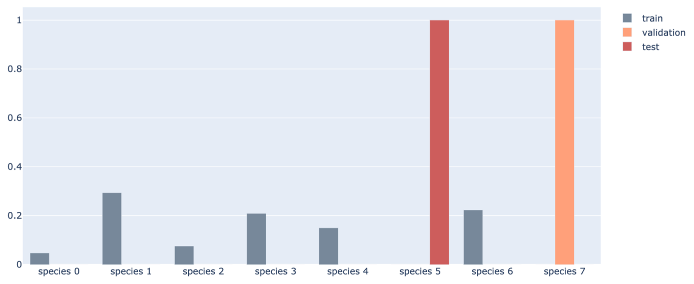


Figure 5: Species information (nodes are proteins from 8 species.)

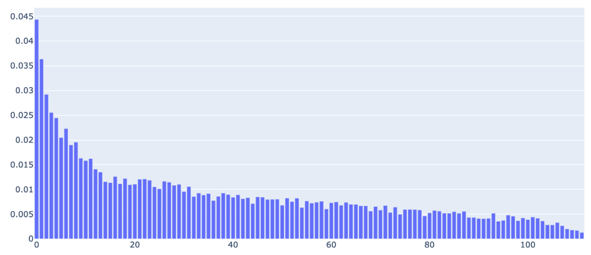


Figure 6: Labels' distribution over training nodes.

5.2. RandCluster-GCN

10 is selected as the number of clusters during training and inference phase as well initially.

Extract Node features. In ClusterGCN work, they prepocessed the node features matrix $\boldsymbol{X}$ by concatenated $\boldsymbol{AX}$ and $\boldsymbol{X}$ to form a new node features matrix $\boldsymbol{X'}$ (as Equation 4 shows) where $\boldsymbol{A}$ is the adjacent matrix. Therefore, exact 1-hop neighborhood for each node and the expensive neighbors searching in the first layer can be saved [5] , which is somehow keeping more information of the whole complete graph before partition the graph into clusters.

$$\boldsymbol{X'} = [\boldsymbol{X}; \boldsymbol{A} \cdot \boldsymbol{X}] \tag{4}$$

We adapt the same idea to save more edges' information for random clustering as **Criterion 1** describes by forming a new feature vector for each node through aggregating the features of edges connected directly, which is defined as follows:

$$\mathbf{x_i} = \Box_{j \in \mathcal{N}(i)} \, \mathbf{e}_{i,j} \tag{5}$$

Where $\Box$ denotes a differentiable, permutation invariant function (e.g. add, mean or max). we extra node features by aggregating the edges' features each node connected directly with three different operations (mean, max and add) [21]respectively. It's proven experimentally that add operation has the best performance (evaluated by ROC-AUC score on the test set). The reason is that add operation could

Aggregation	ROC-AUC
none	0.712 ± 0.0175
add	$\mathbf{0.756 \pm 0.007}$
max	0.745 ± 0.011
mean	0.70 ± 0.018

Table 9: 300 training epochs, best ROC-AUC score on the test set under different aggregation methods (each result is the mean of 10 independent runs plus or minus the standard variance calculated). one-hot node features provided are not used temporarily. None operation means the nodes are initialized by a shared embedding as their features.

always keep the graph structure while the other two may fail in same cases [21].

Do random clustering every epoch. We then apply the concept of **multi-random** mentioned in **Criterion 2** during each training epoch. The result is better in comparison to the **single-random** done before, which is shown in Table 10.

	Aggregation	ROC-AUC
single-random	none	0.712 ± 0.0175
	add	0.756 ± 0.007
multi-random	none	0.717 ± 0.025
	add	$\mathbf{0.765 \pm 0.003}$

Table 10: multi-random improves the best test result obviously.

Decrease the number of clusters during inference. Moreover, We keep the number of clusters unchanged (10) for training and change the number of clusters for inference from the initial 10 to 5 (couldn't be smaller than 4 due to the GPU out of memory issue) according to **Criterion 1**. The latter has better performance as Table 11 shows.

#cluster	Aggregation	ROC-AUC
(10, 10)	none	0.717 ± 0.025
	add	0.765 ± 0.003
(10, 5)	none	0.74 ± 0.0267
	add	$\mathbf{0.767 \pm 0.0052}$

Table 11: multi-random is used as default. About the column of **#cluster**, the first value indicates the number of clusters for training while the second represents that for inference.

Then we combine the node features extracted by add aggregation operation and one-hot node features provided by dataset through concatenation. The best ROC-AUC score is 0.7839 ± 0.0022, which beats the results obtained by Cluster-GCN and node2vec methods whose rankings are 1 and 2 respectively on this dataset for now.

Method	ROC-AUC
Ours	$\mathbf{0.773 \pm 0.0047}$
ClusterGCN-GIN	0.7513 ± 0.0044

Table 12: Our method achieves better performance than ClusterGCN with 300 training epochs.

5.3. Deeper GCNs

Inspired by the success of DeepGCNs work [18], We select a more straightforward techinique which is residual connection to make the original 2-layer GIN model go deeper to improve the model's performance further.

5.3.1 ResGIN

The residual function h in $l+1$ layer is formulated as Equation 6 shows.

$$\mathbf{h}_i^{res(l+1)} = ReLU(\mathbf{\Theta}^{l+1}(\boldsymbol{x}_i^l + \square_{j\in\mathcal{N}(i)}\ (\boldsymbol{x}_j^l + \mathbf{e}_{i,j}))) \quad (6)$$

Where $\square$ denotes a differentiable, permutation invariant function (e.g. add, mean or max, max is used in our case cause it has the best performance based on experimental results. Results shown previously are obtained with default mean operation) and $\mathbf{\Theta}^{l+1}$ represents a MLP of lth layer. Therefore, the output of $l+1$th layer with residual connection is:

$$\mathbf{x}_i^{l+1} = \mathbf{x}_i^l + \mathbf{h}_i^{res(l+1)} \quad (7)$$

We set the number of layers to 3, 7, 14, 28, 56 and 112 respectively, just like DeepGCNs did [18] for ablation study. In addition, the number of training epochs is 1000.

#layers	GIN	ResGIN
3	0.823	0.826
7	0.815	0.826
14	0.815	0.835
28	0.814	0.834
56	0.824	0.8366
112	0.82	0.8227

Table 13: #layers vs. test result (one run) with (ResGIN) and without (GIN) residual connection respectively. It could be observed that ResGIN always outperforms GIN.

5.3.2 Variant ResGIN

Except the residual function defined that is commonly used in GCNs so far (see Equation 6), actually there are variant ways to define residual functions in CNNs, which were fully analyzed in [11] already. We then notice the residual connection used in all GCNs (including ResM-RGCNs) is added after activation function, so called ReLU before addition. Actually, it may not be the best way to add residual connection. As discussed in [11], intuitively the output range of residual function is supposed to be $(-\infty, +\infty)$. Activation function (e.g. ReLU) before addition may impact deep model's representational ability due to the output of residual function is always larger than 0. Therefore, we use full pre-activation [11] to design a more proper residual function for applying residual connection to GIN by changing the sequence of components from $(GINConv \rightarrow Normalization \rightarrow ReLU)$ to $(Normalization \rightarrow ReLU \rightarrow GINConv)$ to satisfy the above output range requirement. We use newResGIN to refer the latter (see Figure 7).

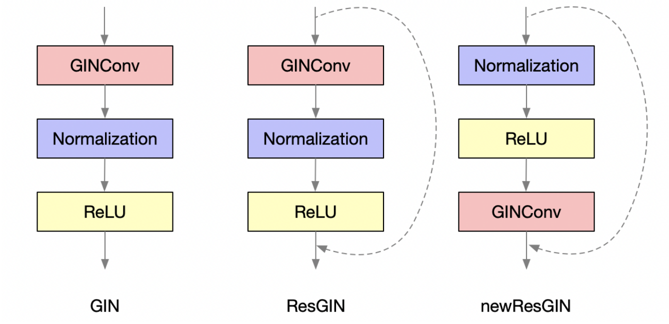


Figure 7: GIN model without residual connection (left: GIN); GIN with addition after $ReLU$ activation (middle: ResGIN); GIN with addition after graph convolution (right: newResGIN). Normalization represents normalization layer (e.g. batch normalization BN [14], layer normalization (LN) [1] etc.). LN outperforms $1d$ BN based on our experimental results.

As Table 14 shows, newResGIN achieves better performance and doesn't have obvious degradation issue when the model goes very deep (112 layers).

#layers	ResGIN	newResGIN
3	0.823 ± 0.0022	$\mathbf{0.8256 \pm 0.0036}$
7	0.83 ± 0.0026	$\mathbf{0.8385 \pm 0.002}$
14	0.832 ± 0.0026	$\mathbf{0.842 \pm 0.0025}$
28	0.835 ± 0.0023	$\mathbf{0.8424 \pm 0.0021}$
56	0.833 ± 0.0057	$\mathbf{0.8457 \pm 0.0033}$
112	0.832 ± 0.0057	$\mathbf{0.844 \pm 0.003}$

Table 14: #layers vs. test result (5 independent runs) of ResGIN and newResGIN respectively with multi-random and 1000 training epochs.

Moreover, because we use mini-batch test based on random clustering (#clsuters=5) as well, a trick could be played here for further improvement, which is another benefit of applying random clustering. Just like doing multi-random during training phase, we can do multi-test by partitioning the graph into 5 clusters several times and get the eventual result by averaging the results obtained and it does improve around 0.5% when the evaluation times is set 5. For fair comparison with the baseline, we don't play this trick here. We compare the best result we get so far with the best result posted on the Leaderboard for ogbn-proteins.

Method	**ROC-AUC**
newResGIN-56	$\mathbf{0.8457 \pm 0.0033}$
GraphSAGE	0.7768 ± 0.0020

Table 15: newResGIN-56 outperforms GraphSAGE by a margin of around 6.89% .

6. Conclusions and Future Work

Except ogbn_proteins this dataset, we also did experiment on another dataset which is ogbn_products. Using random clustering without preprocess we got better performance than ClusterGCN and GraphSAGE these sampling or clustering based methods as well. Although RandClusterGCN just outperformed slightly, it's proven empirically that random clustering is an efficient and effective way to partition large-scale graphs for realizing mini-batch training and reveals its advantage on training very deep GCNs. In the future, we would like to apply random clustering on other datasets and further make the GCNs model go deeper using newResGCN.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[2] Rianne van den Berg, Thomas N Kipf, and Max Welling. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*, 2017.

[3] Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018.

[4] Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 942–950, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[5] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algo-

rithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 257–266, 2019.

[6] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *European conference on computer vision*, pages 628–644. Springer, 2016.

[7] Haoqiang Fan, Hao Su, and Leonidas Guibas. A point set generation network for 3d object reconstruction from a single image. pages 2463–2471, 07 2017.

[8] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034, 2017.

[9] Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. Meshcnn: a network with an edge. *ACM Transactions on Graphics (TOG)*, 38(4):1–12, 2019.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[12] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. Randla-net: Efficient semantic segmentation of large-scale point clouds. *arXiv preprint arXiv:1911.11236*, 2019.

[13] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[14] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[15] Xin Jiang*, Kewei Cheng*, Song Jiang*, and Yizhou Sun. Chordal-{gcn}: Exploiting sparsity in training large-scale graph convolutional networks, 2020.

[16] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

[17] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[18] Guohao Li, Matthias Müller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *The IEEE International Conference on Computer Vision (ICCV)*, 2019.

[19] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images. In *The European Conference on Computer Vision (ECCV)*, September 2018.

[20] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 38(5):1–12, 2019.

[21] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.