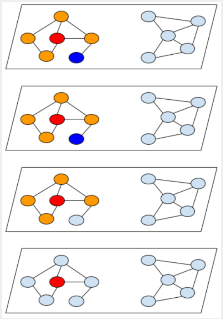
Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks

Paper Authors:
Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, Cho-Jui Hsieh

Student Presenter:
Christopher Risi

University of Waterloo

11 03 2024



Meta-Background

- Published in KDD ’19: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (Click Here)
- Citations as of February 27th 2024:
  - ACM - Digital Library: 568
  - Google Scholar: 1187
- Author Profiles:
  - Wei-Lin Chiang, UC Berkeley
  - Xuanqing Liu, UCLA
  - Si Si, Google Research
  - Yang Li, Google Research
  - Samy Bengio, Apple
  - Cho-Jui Hsieh, UCLA

Timeline of Prior and Derivative Works

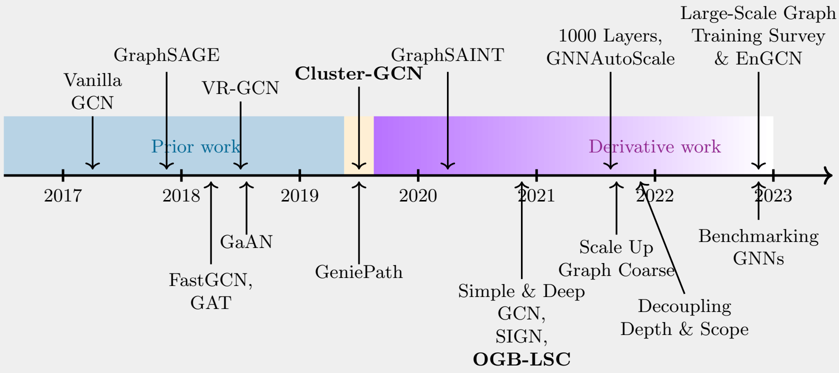


Figure: Timeline of some significant papers, benchmarks, and surveys addressing the scalability of GNNs.

Disclaimer

Please note that some of the figures in these slides is taken directly from Stanford University's CS224WL Machine Learning with Graphs:

Note from Jure Leskovec, Stanford University

Note to other teachers and users of these slides: We would be delighted if you found our material useful for giving your own lectures. Feel free to use these slides verbatim, or to modify them to fit your own needs. If you make use of a significant portion of these slides in your own lecture, please include this message, or a link to our web site: http://cs224w.Stanford.edu

Section 1: Cluster-GCN Background

Brief GCN Background I

We have a graph $G = (\mathcal{V}, \mathcal{E}, A)$ such that:

- $N = |\mathcal{V}|$ vertices.
- $|\mathcal{E}|$ edges.
- A an $N \times N$ sparse adjacency matrix.

Each node is associated with an F-dimensional feature vector and $X \in \mathbb{R}^{N \times F_l}$ is the feature matrix. The embedding for each graph convolutional layer is constructed by:

$$Z^{(l+1)} = A' X^{(l)} W^{(l)}, \quad X^{(l+1)} = \sigma(Z^{(l+1)}),$$

Brief GCN Background II

$$Z^{(l+1)} = A'X^{(l)}W^{(l)}, \quad X^{(l+1)} = \sigma(Z^{(l+1)}),$$

- $X^{(l)} \in \mathbb{R}^{N \times F_l}$ is the embedding at the l-th layer
- $X^{(0)} = X$
- A' is the normalized and regularized adjacency matrix.
- $W^{(l)} \in \mathbb{R}^{F_l \times F_{l+1}}$ the learnable feature transformation matrix.

For a semi-supervised node classification problem, the goal is to learn by minimizing the loss function:

$$\mathcal{L} = \frac{1}{|\mathcal{Y}_L|} \sum_{i \in \mathcal{Y}_L} \text{loss}(y_i, z_i^{(L)}),$$

- $z_i^{(L)}$ is the i-th row of $Z^{(L)}$
- y_i is the ground-truth label.

The *High-Level* Problem I - *Scalable GNNs*

Three high-level **important** problems with previous GCN training algorithms:

1. Memory requirements (scalabilty issues)
2. Time per epoch (training speed)
3. Convergence speed (loss reduction) per epoch (training speed)

CPU:	**GPU**:
Computation: Slow.	**Computation:** Fast.
Memory: 1TB - 10TB	**Memory:** 10GB - 20GB

The *High-Level* Problem II - *Scalable GNNs*

Unlike other neural networks [...], the loss term in GCN depends on a huge number of other nodes, especially when GCN goes deep.

	memory	**time per epoch**	**convergence**
"OG" GCN [8]	bad	good	bad
Mini-batch SGD *GraphSAGE* [5] & *FastGCN* [2]	good	bad	good
VR-GCN [1]	bad	good	good

Table: How do we make all three good? [8] requires storing the entire graph in memory and only gets one weight update per epoch. [2, 5] suffers from the **neighborhood expansion problem**. [1] still requires storing all intermediate embeddings.

Why was Mini-batch SGD Time Per Epoch Bad? - I

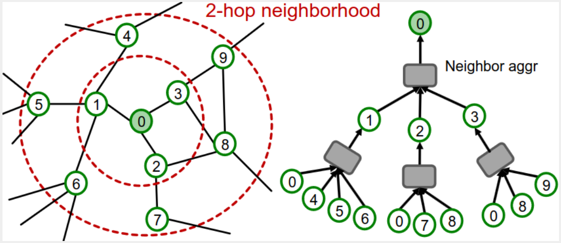
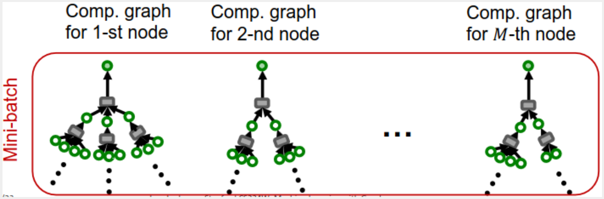


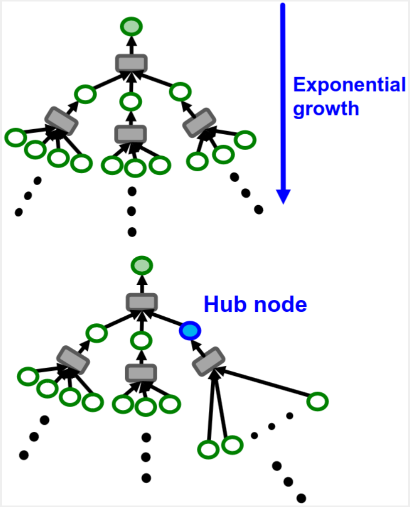
Figure: Neighborhood expansion problem.



Comp. graph for 1-st node
Comp. graph for 2-nd node
Comp. graph for M-th node
Mini-batch
...

Figure: Each batch computes the computation graphs for M nodes.

Why was Mini-batch SGD Time Per Epoch Bad? - II



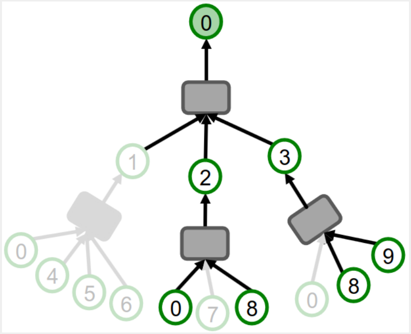
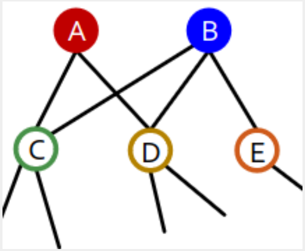


Figure: Even with sampling 2 node embeddings for the computational graph of each node, we get exponential expansion for each added layer.

The *High-Level* Solution - *Scalable GNNs* - I

Notice that we have computational redundancy with our embeddings. How can we take advantage of this?



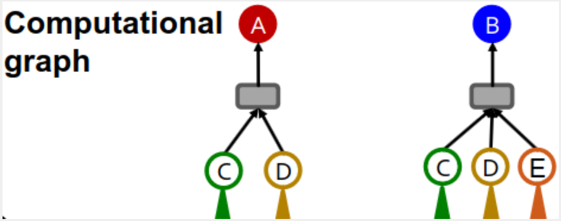
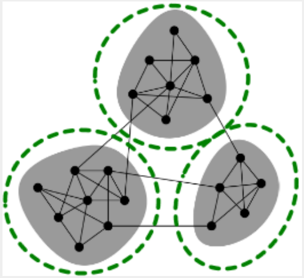


Figure: If *C* and *D* are in the same mini-batch, their encodings can be reused.

The *High-Level* Solution - *Scalable* GNNs - II

If we focus on **maximizing the within-cluster connectivity** and **minimizing the between-cluster connectivity**, we greatly reduce the redundant embedding calculations.



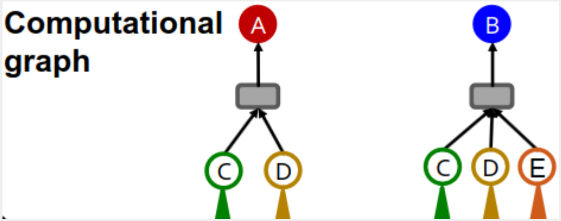


Figure: If *C* and *D* are in the same mini-batch, their encodings can be reused.

The *High-Level* Solution - *Scalable* GNNs - III

The idea behind Cluster-GCN is simple, prior to training, run a clustering algorithm on your graph [7].

- Each mini-batch is assigned all nodes from a single cluster.
- This greatly reduces the neighbourhood expansion problem.

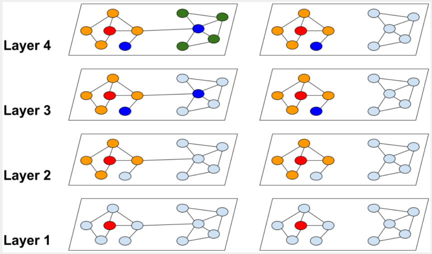


Figure: Left: neighborhood expansion problem. **Right**: Cluster-GCN

The *High-Level* Solution - *Scalable GNNs* - IV

At the time of publication Cluster-GCN[1][3] achieved[2]:

1. Achieved the best memory usage on large-scale graphs
2. A similar training speed with VR-GCN for shallow networks (e.g., 2 layers) but can be faster than VR-GCN when the network goes deeper (e.g., 4 layers),
3. The ability to train a very deep network that has a large embedding size.

[1]Implementation: https://github.com/google-research/google-research/tree/master/cluster_gcn

[2]According to authors of Cluster-GCN

Previous Methods: Complexity Comparison

Complexity	**Time**	**Memory**
GCN [8]	$O(L\|A\|_0 F + LNF^2)$	$O(NLF + LF^2)$
Vanilla SGD	$O(d^L NF^2)$	$O(bd^L F + LF^2)$
GraphSAGE [5]	$O(r^L NF^2)$	$O(br^L F + LF^2)$
FastGCN [2]	$O(rLNF^2)$	$O(brLF + LF^2)$
VR-GCN [1]	$O(L\|A\|_0 F + LNF^2 + r^L NF^2)$	$O(NLF + LF^2)$
Cluster-GCN [3]	$O(L\|A\|_0 F + LNF^2)$	$O(bLF + LF^2)$

- L - the number of layers.
- $\|A\|_0$ - the number of nonzeros in the adjacency matrix.
- F - number of features.
- N - the number of nodes.
- d - average degree of nodes.
- b - batch size.
- r - number of sampled neighbors per node.

Section 2: Cluster-GCN Explained

The *Specific* Problem - Embedding Utilization

This authors clearly outline the problem they are trying to solve in **Section 3.1**:

Problem to Solve

In mini-batch SGD updates, can we design a batch and the corresponding computation subgraph to maximize the **embedding utilization**?

The *Specific* Problem - Embedding Utilization

This authors clearly outline the problem they are trying to solve in **Section 3.1**:

Problem to Solve

In mini-batch SGD updates, can we design a batch and the corresponding computation subgraph to maximize the **embedding utilization**?

Through their work they demonstrate that they can:

Affirmative

We answer this affirmative by connecting the concept of **embedding utilization** to a **clustering objective**.

The *Specific* Problem Continued

This authors clearly outline the problem they are trying to solve in **Section 3.1**:

Problem to Solve

In mini-batch SGD updates, can we design a batch and the corresponding computation subgraph to maximize the **embedding utilization**?

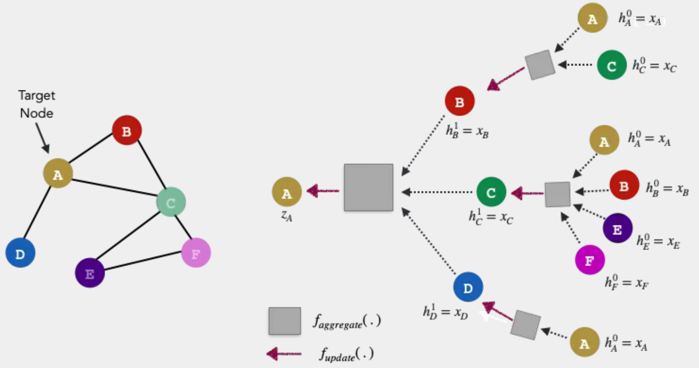
Definition

Embedding Utilization: During the algorithm, if the node i's embedding at the l-th layer z_i^l is computed and is reused u times for the embedding computations at layer $l+1$, then we say the embedding utilization of z_i^l is u.

Why is embedding utilization important? - I

The computation of one node i : $\nabla \text{loss}(y_i, z_i^{(L)})$ for GraphSAGE:

- If a GCN has $L+1$ layers, and each node has avg. degree d:
  - We aggregate features for $O(d^L)$ nodes in the graph for i.
  - Each node embedding requires $O(F^2)$ time.
  - Average computing of the gradient for i requires $O(d^L F^2)$



Why is embedding utilization important? - II

Now consider more than one node.
In the worst case the complexity is $O(bd^L)$, however different nodes can have overlapping k-hop neighbors, the greater the overlap, the greater the **embedding utilization.**

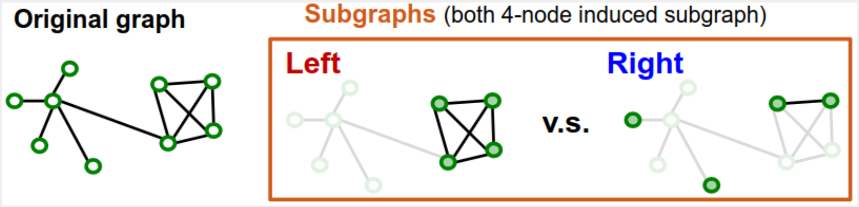


Figure: Left: high embedding utilization. **Right**: low embedding utilization.

Why is embedding utilization important? - III

If the **embedding utilization** u is small, then SGD needs:

- $O(bd^L)$ embeddings per batch
- $O(bd^L F^2)$ time per update
- $O(Nd^L F^2)$ per epoch

Consider a set of nodes $\mathcal{B}$ from layer 1 to L. The same subgraph $A_{\mathcal{B},\mathcal{B}}$ is used for each layer of computation.

Why is embedding utilization important? - IV

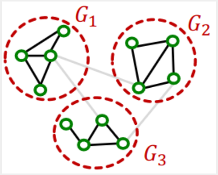
Consider a set of nodes $\mathcal{B}$ from layer 1 to L. The same subgraph $A_{\mathcal{B},\mathcal{B}}$ is used for each layer of computation.

- the **embedding utilization** for this subgraph is $\|A_{\mathcal{B},\mathcal{B}}\|_0$.
- to maximize **embedding utilization** we must *maximize the number of within-batch edges*.

This is the entire motivation for this paper!

Cluster-GCN Explained

$$\mathcal{V} = [\mathcal{V}_1, \cdots, \mathcal{V}_c]$$
$$\bar{G} = [G_1, \cdots, G_c] = [\{\mathcal{V}_1, \mathcal{E}_1\}, \cdots, \{\mathcal{V}_c, \mathcal{E}_c\}]$$



$$A = \bar{A} + \Delta$$

$$\begin{bmatrix} A_{11} & \dots & A_{1c} \\ \vdots & \ddots & \vdots \\ A_{c1} & \dots & A_{cc} \end{bmatrix} = \begin{bmatrix} A_{11} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & A_{cc} \end{bmatrix} + \begin{bmatrix} 0 & \dots & A_{1c} \\ \vdots & \ddots & \vdots \\ A_{c1} & \dots & 0 \end{bmatrix}$$

The loss function becomes:

$$\mathcal{L}_{\bar{A}'} = \sum_t \frac{|\mathcal{V}_t|}{N} \mathcal{L}_{\bar{A}'_{tt}}$$

$$\mathcal{L}_{\bar{A}'} = \sum_t \frac{|\mathcal{V}_t|}{N} \left(\frac{1}{|\mathcal{V}_t|} \sum_{i \in \mathcal{V}_t} \text{loss}(y_i, z_i^{(L)}) \right)$$

Cluster-GCN Algorithm

Algorithm 1 Cluster GCN

```
Input: Graph $A$, feature $X$, label $Y$;
Output: Node representation $\bar{X}$
1: Partition graph nodes in $c$ clusters $\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_c$ by METIS:
2: for $iter = 1, \cdots, max\_iter$ do
3:     Randomly choose $q$ clusters, $t_1, \cdots, t_q$ from $\mathcal{V}$ without replacement;
4:     Form the subgraph $\bar{G}$ with nodes $\bar{\mathcal{V}} = [\mathcal{V}_{t_1}, \mathcal{V}_{t_2}, \cdots, \mathcal{V}_{t_q}]$ and links $A_{\bar{\mathcal{V}},\bar{\mathcal{V}}}$;
5:     Compute $g \leftarrow \nabla \mathcal{L}_{A_{\bar{\mathcal{V}},\bar{\mathcal{V}}}}$ (loss on the subgraph $A_{\bar{\mathcal{V}},\bar{\mathcal{V}}}$);
6:     Conduct Adam update using gradient estimator $g$;
7: end for
8: Output: $\{W_l\}_{l=1}^{L}$
```

Cluster-GCN *One more thing to consider*

Algorithm 2 Cluster GCN

```
Input: Graph $A$, feature $X$, label $Y$;
Output: Node representation $\bar{X}$
1: Partition graph nodes in $c$ clusters $\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_c$ by METIS:
2: for $iter = 1, \cdots, max\_iter$ do
3:     Randomly choose $q$ clusters, $t_1, \cdots, t_q$ from $\mathcal{V}$ without replacement;
4:     Form the subgraph $\bar{G}$ with nodes $\bar{\mathcal{V}} = [\mathcal{V}_{t_1}, \mathcal{V}_{t_2}, \cdots, \mathcal{V}_{t_q}]$ and links $A_{\bar{\mathcal{V}},\bar{\mathcal{V}}}$;
5:     Compute $g \leftarrow \nabla \mathcal{L}_{A_{\bar{\mathcal{V}},\bar{\mathcal{V}}}}$ (loss on the subgraph $A_{\bar{\mathcal{V}},\bar{\mathcal{V}}}$);
6:     Conduct Adam update using gradient estimator $g$;
7: end for
8: Output: $\{W_l\}_{l=1}^{L}$
```

Cluster-GCN Stochastic Multiple Partitions

Issues arrise if we only choosing one cluster per batch, i.e. $q = 1$.

- After the graph is partitioned, some links (the Δ part) are removed.
- Graph clustering algorithms tend to **bring similar nodes together**.
- Therefore, the distribution of a cluster could be different from the full data set, leading to a **biased estimation of the full gradient**.

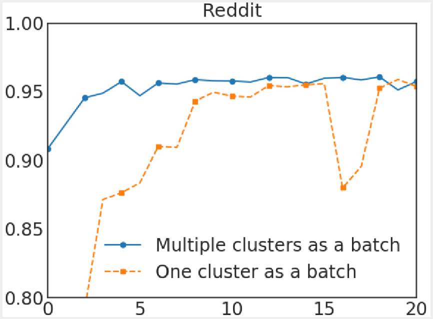


Figure: The $q = 1$ uses 300 partitions. The $q = 5$ uses 1500 partitions. Epoch (x-axis) versus F1 score (y-axis).

Cluster-GCN Stochastic Multiple Partitions

This is what the multiple partitions looks like:

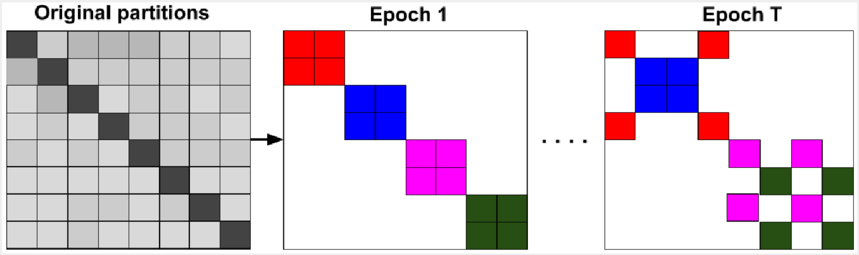


Figure: $q = 2$ and $c = 8$, the diagonals represent $\bar{A}$, and off diagonal represents Δ. Each diagonal square represents a subgraph A_{tt}.

Cluster-GCN One *final* deep thing to consider

In [8] for deeper GCNs they suggest adding a residual connection from the previous layer $X^{(l)}$:

$$X^{(l+1)} = \sigma(A' X^{(l)} W^{(l)}) + X^{(l)}$$

[3] instead thematically recommends amplifying the diagonal parts of the of A, considering:

$$X^{(l+1)} = \sigma((A' + I) X^{(l)} W^{(l)})$$

This may not be suitable and suffer from numerical instability, further modifying it maintains neighborhood information and numerical ranges:

$$\tilde{A} = (D + I)^{-1}(A + I)$$

and

$$X^{(l+1)} = \sigma((\tilde{A} + \lambda \mathrm{diag}(\tilde{A})) X^{(l)} W^{(l)})$$

Cluster-GCN *Summary*

Summary

- During pre-training, a graph clustering algorithm (e.g. *METIS*[7]) partitions the graph into subgraphs maximizing within cluster connectivity.
- For each batch, q clusters are sampled with the inter-cluster edges included.
- The layer-wise node embeddings are calculated for the batch subgraph.
- A full epoch is when all clusters have been sampled once without replacement.

Cluster-GCN *Pros & Cons*

Pro

Compared to previous methods, Cluster-GCN tends to improve efficiency (especially when more layers are useful):

- memory: good; time per epoch: good; convergence: good

Caution

A potential concern is when training significantly benefits from aggregations between clusters and/or long distance nodes.

Section 3: Experimental Results

Experiments

The authors broke their experimentation into three parts:

1. Training performance for medium size datasets
   - Training Time vs Accuracy
   - Memory usage comparisons
2. Experimental results on Amazon2M (large dataset)
3. Training deeper GCN

The Hardware: Then vs Now

- All the experiments in 2019 were conducted on a machine with a NVIDIA Tesla V100 GPU (16 GB memory), 20-core Intel Xeon CPU (2.20 GHz), and 192 GB of RAM.
- Contrasting this with some of the hardware used on the OGB leaderboards (March 2024):
  - **MAG240M**: V100 (15-32GB), A100 80GB, 4 Google Cloud TPUv4
  - **WikiKG90Mv2**: Graphcore Bow Pod16, 2 TitanV, 4 Tesla M40, A100
  - **PCQM4Mv2**: 32 V100s (32GB), 8 A100s, Graphcore BOW-POD16

The Data & Experimental Parameters

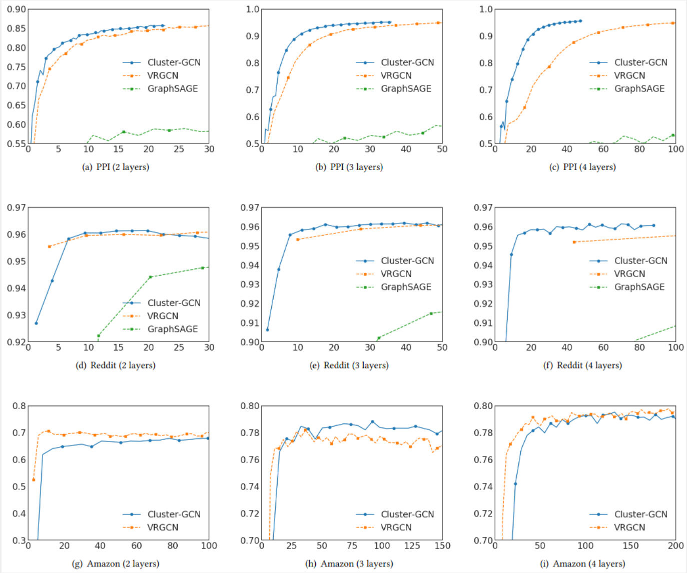
Datasets	Task	#Nodes	#Edges	#Labels	#Features
PPI	multi-label	56,944	818,716	121	50
Reddit	multi-label	232,965	11,606,919	41	602
Amazon	multi-label	344,863	925,872	58	N/A
Amazon2M	multi-class	2,449,029	61,859,140	47	100

Table: Data statistics. Large size dataset.

Datasets	#hidden units	#partitions	#clusters per batch
PPI	512	50	1
Reddit	128	1500	20
Amazon	128	200	1
Amazon2M	400	15000	10

Table: The parameters used in the experiments. Large size dataset.

Training for Medium Sized Datasets: *Training Time vs Accuracy*

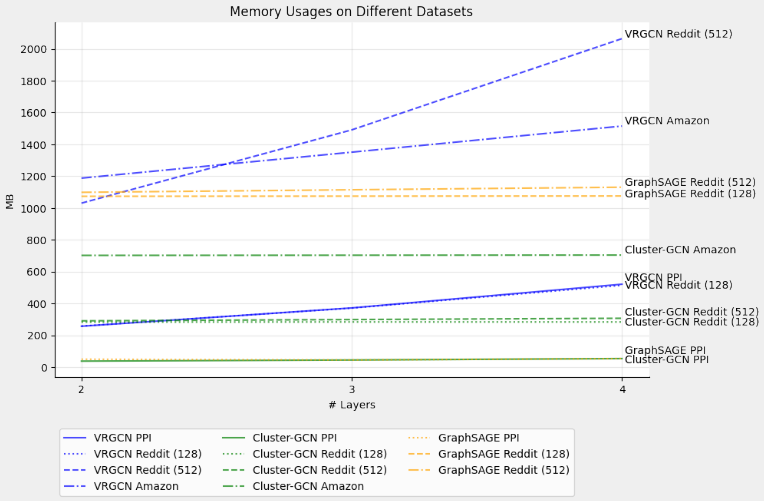


(a) PPI (2 layers) (b) PPI (3 layers) (c) PPI (4 layers)

(d) Reddit (2 layers) (e) Reddit (3 layers) (f) Reddit (4 layers)

(g) Amazon (2 layers) (h) Amazon (3 layers) (i) Amazon (4 layers)

Training for Medium Sized Datasets: *Memory Usage Comparisons*



Training for Large Sized Datasets: *Amazon2M Dataset*

	Time		**Memory**		**Test F1 score**	
	VRGCN	Cluster-GCN	VRGCN	Cluster-GCN	VRGCN	Cluster-GCN
Amazon2M (2L)	337 s	1223 s	7476 MB	2228 MB	89.03	89.00
Amazon2M (3L)	1961 s	1523 s	11218 MB	2235 MB	90.21	90.21
Amazon2M (4L)	N/A	2289 s	OOM	2241 MB	N/A	90.41

Table: Comparisons of running time, memory and testing accuracy (F1 score) for Amazon2M.

Training Deeper GCNs - I

	2-layer	3-layer	4-layer	5-layer	6-layer
Cluster-GCN	52.9 s	82.5 s	109.4 s	137.8 s	157.3 s
VRGCN	103.6 s	229.0 s	521.2 s	1054 s	1956 s

Table: Comparisons of running time when using different numbers of GCN layers. The authors use PPI and run both methods for 200 epochs.

- Cluster-GCN requires less additional time per layer.
- VRGCN is almost doubling with each additional layer.

Training Deeper GCNs - II

1. $Z^{(l+1)} = A'X^{(l)}W^{(l)}, \quad X^{(l+1)} = \sigma(Z^{(l+1)})$
2. $X^{(l+1)} = \sigma((A' + I)X^{(l)}W^{(l)})$
3. $\tilde{A} = (D + I)^{-1}(A + I)$
4. $X^{(l+1)} = \sigma((\tilde{A} + \lambda\text{diag}(\tilde{A}))X^{(l)}W^{(l)})$

	2-layer	3-layer	4-layer	5-layer	6-layer	7-layer	8-layer
Cluster-GCN w (1)	90.3	97.6	98.2	98.3	94.1	65.4	43.1
Cluster-GCN w (3)	90.2	97.7	98.1	98.4	42.4	42.4	42.4
Cluster-GCN w (3) + (2)	84.9	96.0	97.1	97.6	97.3	43.9	43.8
Cluster-GCN w (3) + (4), $\lambda = 1$	89.6	97.5	98.2	98.3	98.0	97.4	96.2

Table: Comparisons of using different diagonal enhancement techniques. For all methods, we present the best validation accuracy achieved in 200 epochs. PPI is used and dropout rate is 0.1 in this experiment.

Training Deeper GCNs - III

Cluster-GCN with the diagonal enhancement techniques allow for much deeper GCNs that out performed the other state of the art methods (at the time):

	PPI	Reddit
FastGCN	N/A	93.7
GraphSAGE	61.2	95.4
VR-GCN	97.8	96.3
GaAN	98.71	96.36
GAT	97.3	N/A
GeniePath	98.5	N/A
Cluster-GCN	**99.36**	**96.60**

The Influenced:

Problems to address and things to think about:

- How do you know what the right q is? How is the overhead addressed?
- ClusterGCN does not take into account pre-processing time, for large enough graphs clustering preprocessing can exceed training time (GraphSAINT)[10].
- Even with these new methods, it does not scale well with increasing number of layers (1000-layers)[9].
- Most **sampling-based techniques** (GraphSAGE, FastGCN, Cluster-GCN, GraphSAINT) appear to be surpassed by the better (not all) **decoupling-based techniques** (GAMLP, C&S)[11, 6] and ensemble techniques (enGCN)[4].
  - decoupling is when propagation ($A^{(l)}X^{(l)}$) and prediction ($X^{(l)}W^{(l)}$) are performed separately[4].

References I

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. PMLR.

Jie Chen, Tengfei Ma, and Cao Xiao.
FastGCN: Fast learning with graph convolutional networks via importance sampling.
In *6th International Conference on Learning Representations*, ICLR'18. OpenReview.net.

Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh.
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*, KDD '19, pages 257–266. Association for Computing Machinery.

Keyu Duan, Zirui Liu, Peihao Wang, Wenqing Zheng, Kaixiong Zhou, Tianlong Chen, Xia Hu, and Zhangyang Wang.
A comprehensive study on large-scale graph training: Benchmarking and rethinking.
In *36th Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

William L. Hamilton, Rex Ying, and Jure Leskovec.
Inductive representation learning on large graphs.
In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 1025–1035. Curran Associates Inc.
event-place: Long Beach, California, USA.

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

References II

George Karypis and Vipin Kumar.
A fast and high quality multilevel scheme for partitioning irregular graphs.
20(1):359–392.
Publisher: Society for Industrial and Applied Mathematics.

Thomas N. Kipf and Max Welling.
Semi-supervised classification with graph convolutional networks.
In *5th International Conference on Learning Representations*, ICLR'17. OpenReview.net.

Guohao Li, Matthias Müller, Bernard Ghanem, and Vladlen Koltun.
Training graph neural networks with 1000 layers.
In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 6437–6449. PMLR.

Hanqing Zeng, Ajitesh Srivastava, Viktor Prasanna, Hongkuan Zhou, and Rajgopal Kannan.
GraphSAINT: Graph sampling based inductive.

Wentao Zhang, Ziqi Yin, Zeang Sheng, Yang Li, Wen Ouyang, Xiaosen Li, Yangyu Tao, Zhi Yang, and Bin Cui.
Graph attention multi-layer perceptron.
In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 4560–4570.