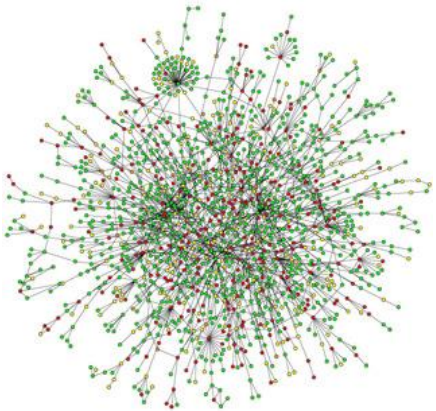
Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks

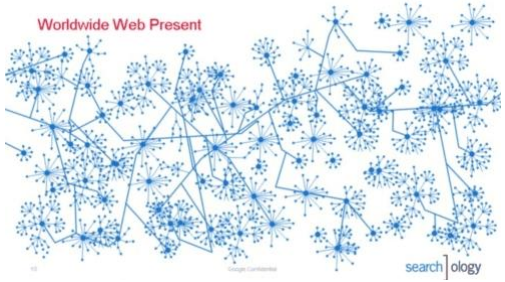
Wei-Lin Chiang[1], Xuanqing Liu[2], Si Si[3], Yang Li[3], Samy Bengio[3], Cho-Jui Hsieh[23]
[1]National Taiwan University, [2]UCLA, [3]Google Research

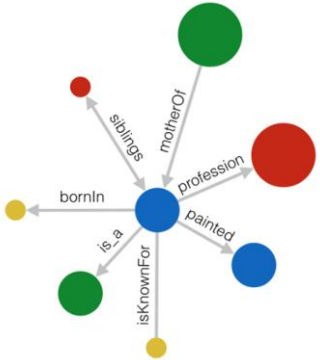
國立臺灣大學 National Taiwan University UCLA Google AI

Graph Convolutional Networks



- GCN has been successfully applied to many graph-based applications
- For example, social networks, knowledge graphs and biological networks
- However, training a large-scale GCN remains challenging



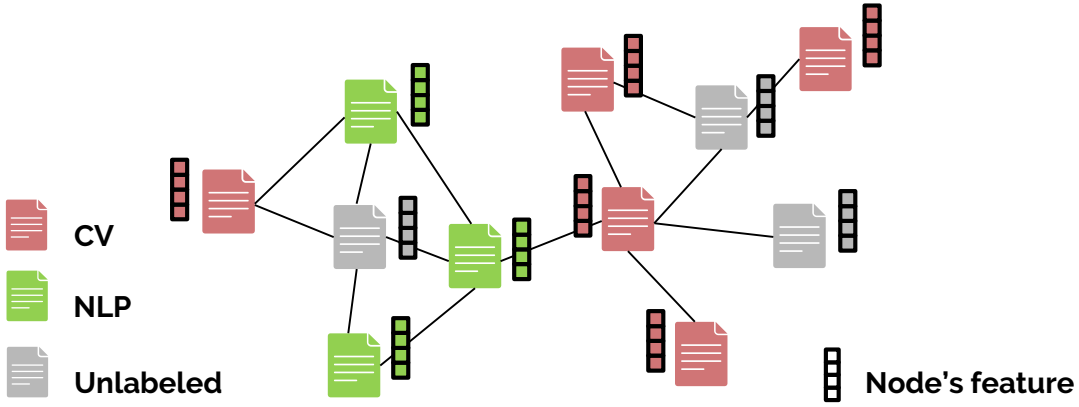


Leonardo da Vinci

Background of GCN

Let's start with an example of citation networks

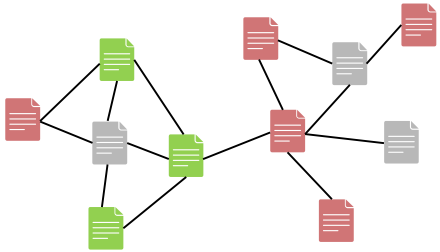
- Node: paper, Edge: citation, Label: category
- Goal: predict the unlabeled ones (grey nodes)



Notations

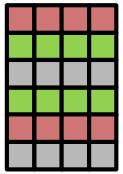
Adjacency matrix: $\boldsymbol{A}$
$(N - by - N$ matrix$)$

$$\begin{bmatrix} 0 & 1 & \cdots & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ \vdots & 1 & \ddots & 0 & \vdots \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & \cdots & 0 & 0 \end{bmatrix}$$



Feature matrix: $\boldsymbol{X}$
$(N - by -$ F matrix$)$

$$\begin{bmatrix} 0 & 0.3 & \cdots & 0.8 & 0.9 \\ 0.4 & 0 & 0.6 & 0.1 & 0 \\ \vdots & 0.2 & \ddots & 0 & \vdots \\ 0 & 0.5 & 0 & 0 & 0 \\ 0.3 & 0.2 & \cdots & 0 & 0 \end{bmatrix}$$



Label vector: $\boldsymbol{Y}$

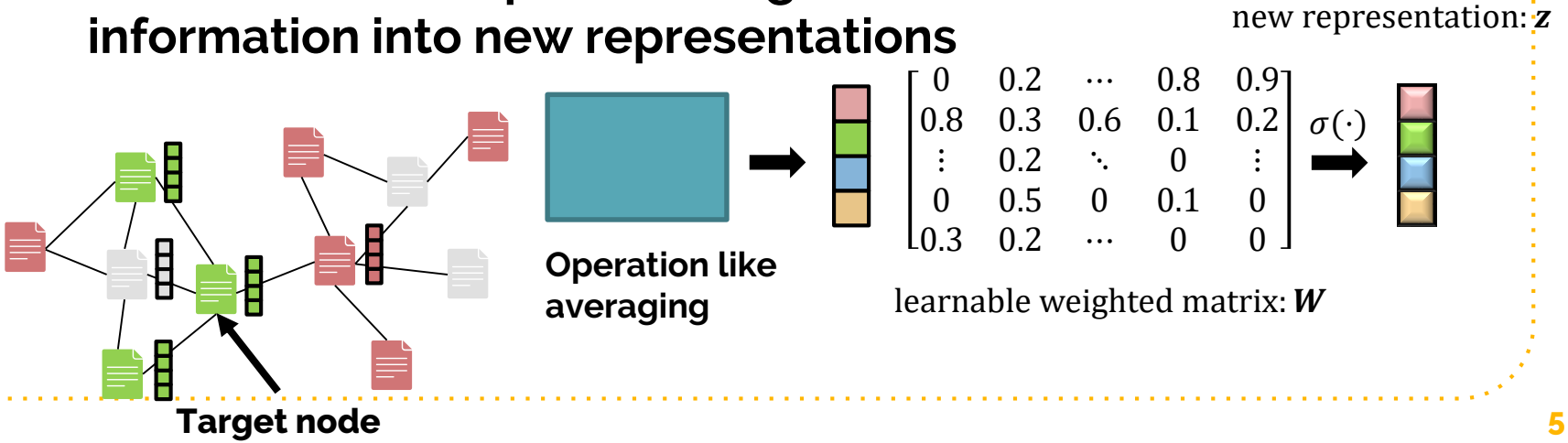
$$[0 \quad 1 \quad \cdots \quad 0 \quad 1]^T$$

A GCN Update

- In each GCN layer, node's representation is updated through the formula:

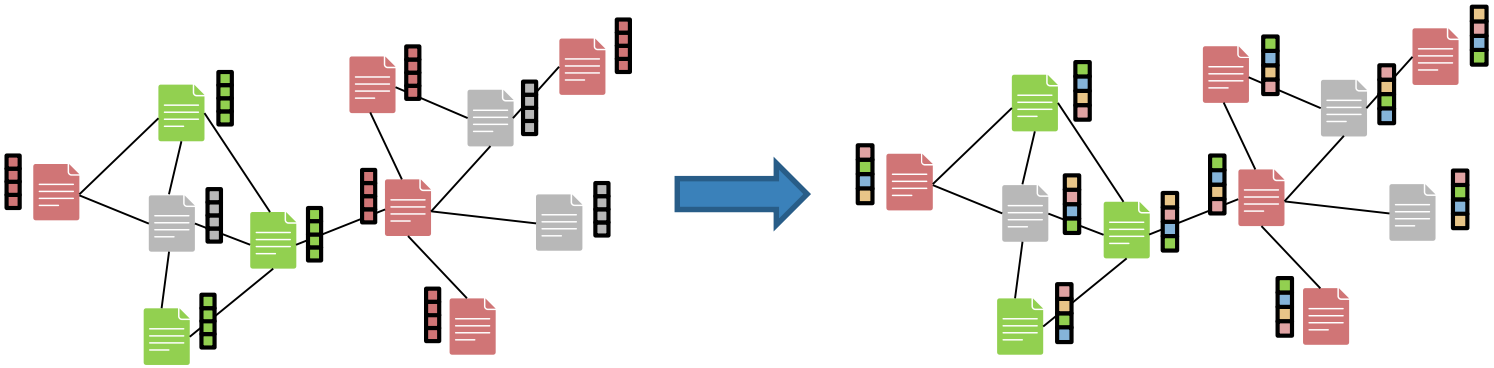
$$X^{(l+1)} = \sigma(AX^{(l)}W^{(l)})$$

- The formula incorporates neighborhood information into new representations



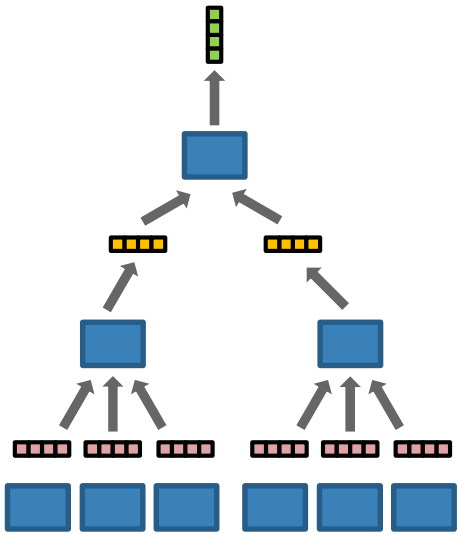
Better Representations

- After GCN update, we hope to obtain better node representations aware of local neighborhoods
- The representations are useful for downstream tasks



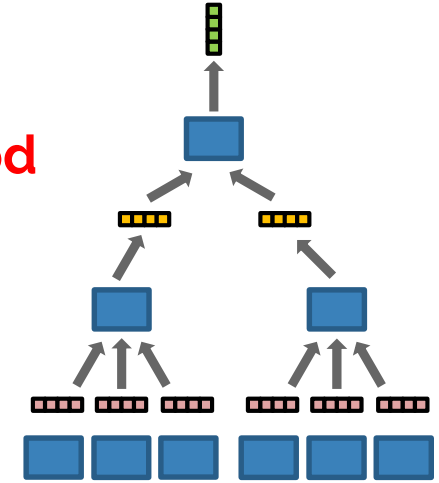
But Training GCN is not trivial

- In standard neural networks (e.g., CNN), loss function can be decomposed as $\sum_{i=0}^{N} \boldsymbol{loss}(x_i, y_i)$
- However, in GCN, loss on a node not only depends on itself but all its neighbors
- This dependency brings difficulties when performing SGD on GCN



What's the Problem in SGD?

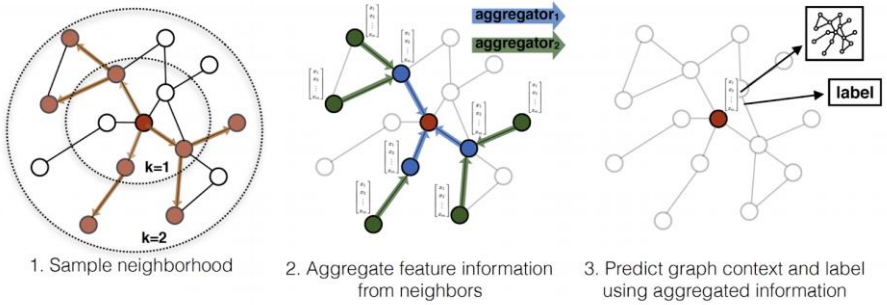
- Issues come from high computation costs
- Suppose we desire to calculate a target node's loss with a 2-layer GCN
- To obtain its final representation, needs all node embeddings in its 2-hop neighborhood
- 9 nodes' embeddings needed but only get 1 loss (utilization: low)



How to Make SGD Efficient for GCN?

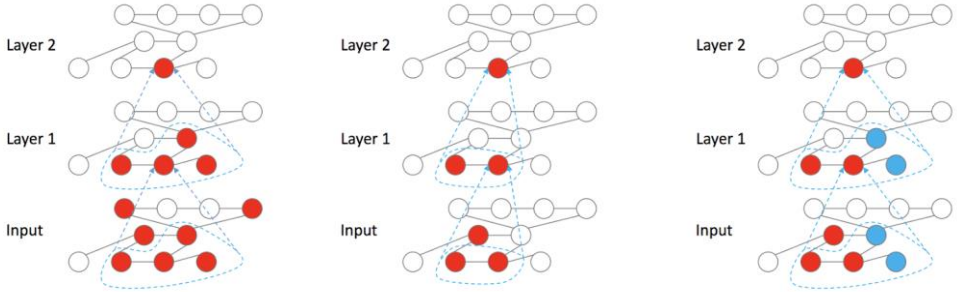
Idea: subsample a smaller number of neighbors

- For example, GraphSAGE (NeurIPS'17) considers a subset of neighbors per node
- But it still suffers from recursive neighborhood expansion



How to Make SGD Efficient for GCN?

- VRGCN (ICML'18) subsamples neighbors and adopts variance reduction for better estimation
- But it introduces extra memory requirement (#node x #feature x #layer)

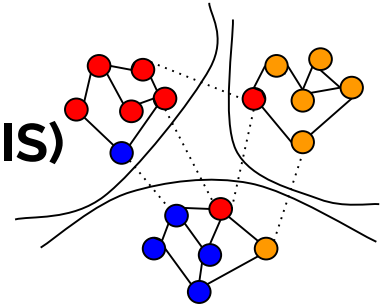


(a) Exact (b) Neighbour sampling (c) Control variate

Improve the Embedding Utilization

- If considering all losses at one time (full-batch),
  $$GCN_{2-layer}(A, X) = A\sigma\left(AXW^{(0)}\right)W^{(1)},$$
  9 nodes' embedding used and got 9 losses
- Embedding Utilization: optimal
- The key is to re-use nodes' embeddings as many as possible
- Idea: focus on dense parts of the graph

Graph Clustering Can Help!

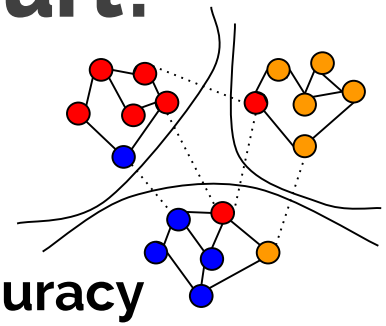


Idea: apply graph clustering algorithm (e.g., METIS) to identify dense subgraphs.

Our proposed method: Cluster-GCN

- Partition the graph into several clusters, remove between-cluster edges
- Each subgraph is used as a mini-batch in SGD
- Embedding utilization is optimal because nodes' neighbors stay within the cluster

Issue: Does Removing Edges Hurt?

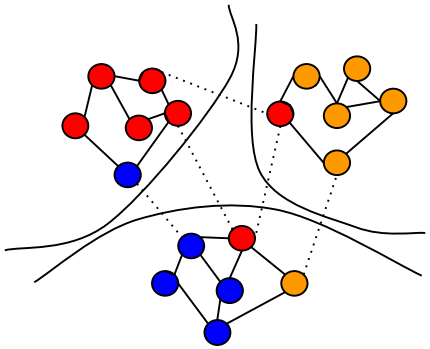


- An example on CiteSeer
  (a citation network with 3327 nodes)
- Even though 20% edges are removed, the accuracy of GCN model remains similar

CiteSeer	Random partitioning	Graph partitioning
1 (no partitioning)	72.0	72.0
100 partitions	46.1	71.5 (~20% edges removed)

Issue: imbalanced label distribution

- However, nodes with similar labels are clustered together
- Hence the label distribution within a cluster could be different from the original data
- Leading to a biased SGD!

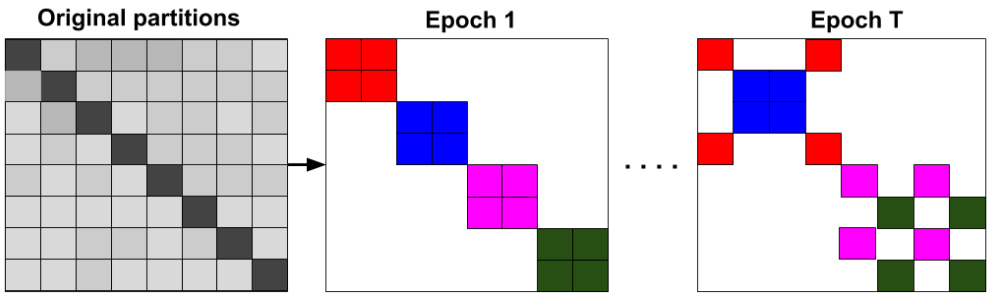


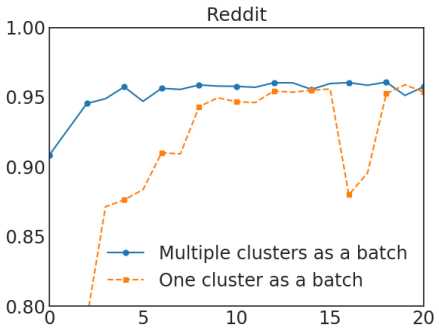
Selection of Multiple Clusters

We propose to randomly select multiple clusters as a batch.

Two advantages:

- Balance label distribution within a batch
- Recover some missing edges between-cluster





Experiment Setup

- **Cluster-GCN:**
  METIS as the graph clustering method
- **GraphSAGE (NeurIPS'17):**
  samples a subset of neighbors per node
- **VRGCN (ICML'18)**
  subsample neighbors + variance reduction

Datasets

- Reddit is the largest public data in previous papers
- To test scalability, we construct a new data Amazon2M (2 million nodes) from Amazon co-purchasing product networks

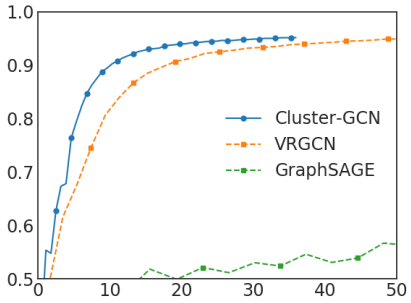
Datasets	Task	#Nodes	#Edges	#Labels	#Features
PPI	multi-label	56,944	818,716	121	50
Reddit	multi-class	232,965	11,606,919	41	602
Amazon	multi-label	334,863	925,872	58	N/A
Amazon2M	multi-class	2,449,029	61,859,140	47	100

Comparisons on Medium-size Data

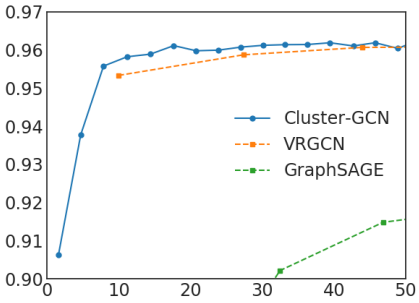
We consider a 3-layer GCN.

(X-axis: running time in sec, Y-axis: validation F1)

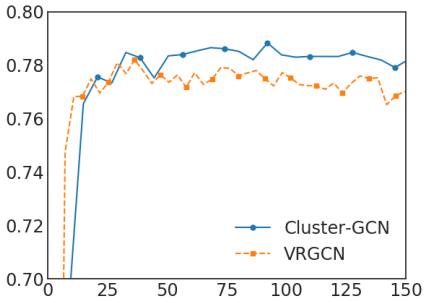
- GraphSAGE is slower due to sampling many neighbors
- VRGCN, Cluster-GCN finish the training in 1 minute for those three data



PPI



Reddit



Amazon (GraphSAGE OOM)

Comparisons on #GCN-Layers

- Cluster-GCN is suitable for deeper GCN training
- The running time of VRGCN grows exponentially with #GCN-layer, while Cluster-GCN grows linearly

Table 9: Comparisons of running time when using different numbers of GCN layers. We use PPI and run both methods for 200 epochs.

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

Comparisons on Million-scale Graph

- Amazon2M: 2M nodes, 60M edges and only a single GPU used
- VRGCN encounters memory issue while using more GCN layers (due to VR technique)
- Cluster-GCN is scalable to million-scale graphs with less and stable memory usage

Table 8: Comparisons of running time, memory and testing accuracy (F1 score) for Amazon2M.

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

Is Deep GCN Useful?

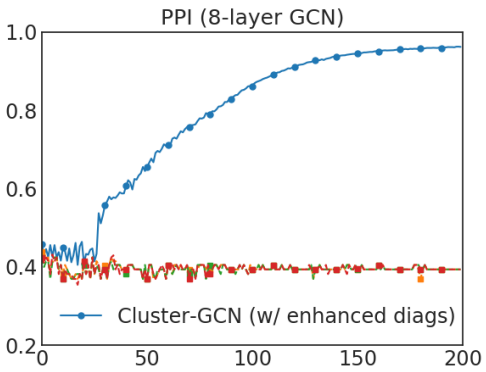
- Consider a 8-layer GCN on PPI

$$Z = \mathrm{softmax}\left(A \cdots \sigma\left(A\sigma\left(AXW^{(0)}\right)W^{(1)}\right) \cdots W^{(7)}\right)$$

- Unfortunately, existing methods fail to converge
- To facilitate training, we develop a useful technique, "diagonal enhancement"

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

- Cluster-GCN finishes 8-layer GCN training in only few minutes



(X-axis: running time, Y-axis: validation F1)

Cluster-GCN achieves SoTA

- With deeper & wider GCN, SoTA results achieved
- PPI: 5-layer GCN with 2048 hidden units
- Reddit: 4-layer GCN with 128 hidden units

Table 10: State-of-the-art performance of testing accuracy reported in recent papers.

	PPI	Reddit
FastGCN [1]	N/A	93.7
GraphSAGE [5]	61.2	95.4
VR-GCN [2]	97.8	96.3
GaAN [16]	98.71	96.36
GAT [14]	97.3	N/A
GeniePath [10]	98.5	N/A
Cluster-GCN	**99.36**	**96.60**

Conclusions

In this work, we propose a simple and efficient training algorithm for large and deep GCN.

- Scalable to million-scale graphs
- Allow training on deeper & wider GCN models
- Achieve state-of-the-art on public data
- TensorFlow codes available at https://github.com/google-research/google-research/tree/master/cluster_gcn