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], and Cho-Jui Hsieh[23]

[1]National Taiwan University, [2]UCLA, [3]Google Research

Introduction

- Training large-scale GCN is a challenging problem. Existing SGD-based methods suffer from either a high computational cost or a large memory requirement
- In this work,
  - ➢ We propose Cluster-GCN which exploits graph clustering structure and is suitable for SGD-based training on GCN
  - ➢ With Cluster-GCN, large (million-scale) and deep GCN training is possible, leading to SoTA results on public data sets (e.g., PPI, Reddit)

Graph Convolutional Networks

- $A \in \mathbb{R}^{N\times N}$: adjacency matrix, $X \in \mathbb{R}^{N\times F}$: feature matrix, $Y \in \mathbb{R}^{N}$: label vector, $W^{(l)} \in \mathbb{R}^{F_l \times F_{l+1}}$: learnable weight matrices, $\sigma(\cdot)$: activation function
- In each GCN layer, nodes' representations are updated through the formula:

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

- Neighborhood information is incorporated into each node's representation, useful for downstream tasks (e.g., node classification, link prediction)

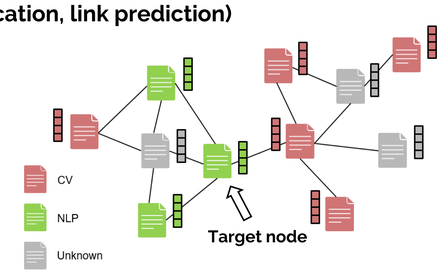


Figure 1. An example of citation networks.

What's Special in GCN?

- In standard neural networks (e.g., CNN), loss function can be decomposed as $\sum_{i=1}^{N} loss(x_i, y_i)$
- However, in GCN, loss on a node not only depends on itself but all its neighbors. This brings difficulties when performing SGD on GCN

Embedding Utilization (EU)

In Figure 1, suppose a 2-layer GCN is used and we focus only the targeted node's loss. To obtain its final node representation, we need all nodes' embeddings in its 2-hop neighborhood.

- 10 embeddings used but only get 1 loss (EU: low)
- Same situation occurs in SGD on large and sparse graph. Plus, #neighbors can grow exponentially (neighborhood explosion) with more #GCN-layers

By contrast, if computing all losses at one time, 11 embedding used and 11 losses calculated (EU: optimal)

$$GCN_{2-layer}(A, X) = A\sigma\left(AXW^{(0)}\right)W^{(1)}$$

- The key is to re-use nodes' embeddings as many as possible (e.g., use full-batch)
- But a full-batch gradient descent may be inferior from the perspective of optimization (e.g., convergence rate)

Efficient SGD Methods for GCN?

Subsample smaller #neighbors for each node (denoted: r):

- GraphSAGE (NeurIPS'17) samples a fixed number of neighbors
- VRGCN (ICML'18) adopts a small r per node + variance reduction technique for better estimation
- FastGCN (ICLR'18) fixes r per layer + importance sampling

But several issues still exist:

1. Recursive neighborhood expansion (low EU)
2. Objective estimation is not as good as before
3. Extra memory requirements (VR technique)

Our Proposed Method: Cluster-GCN

Idea: use graph clustering to capture important local information and improves efficiency

- Example: CiteSeer (a citation network with 3327 nodes)
- If applying graph partitioning (e.g., metis) on A and removed between-cluster edges, the accuracy of GCN remains similar (even though 20% edges are removed)

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

Cluster-GCN:

- ➢ Partition the original graph into several clusters
- ➢ Each cluster (subgraph) can be seen as a mini-batch in SGD
- ➢ Neighbors of cluster nodes keep within the cluster. EU is optimal

Robust training for Cluster-GCN

Issue: nodes with similar labels tend to be clustered.
Hence the label distribution within a mini-batch could be different from original data, leading to a biased gradient estimation in SGD!
To alleviate the imbalance issue, we randomly select multiple clusters as a batch. Two advantages:

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

Algorithm 1: Cluster GCN

Input: Graph A, feature X, label Y;
Output: Node representation $\bar{X}$

1 Partition graph nodes into 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 Output: $\{W_l\}_{l=1}^{L}$

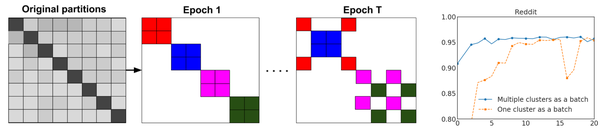


Figure 2. Explanation of multiple clusters selection & results

Experiments

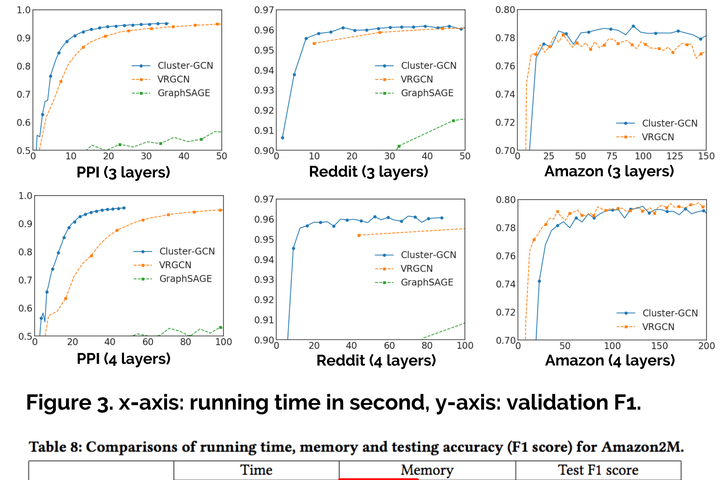


Figure 3. x-axis: running time in second, y-axis: validation F1.

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

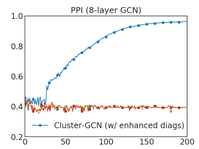
Comparisons on Deeper GCN

- The running time of VRGCN grows exponentially with #GCN-layers, while Cluster-GCN grows linearly

	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

- Training deep GCN is difficult: 8-layer GCN on PPI fails to converge
- We develop a technique "diagonal enhancement"

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



SoTA results through deeper & wider GCN

- PPI: 5-layer with 2048 hidden units
- Reddit: 4-layer with 128 hidden units

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

	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**

Table 1. SoTA results from recent papers.

Conclusions

We present a fast and memory efficient GCN training algorithm, Cluster-GCN, which can train on large-scale graphs with over 2 million nodes in less than an hour. Cluster-GCN also allows deeper and wider GCN, leading to SoTA performance on PPI and Reddit datasets.
TensorFlow implementation available at: https://github.com/google-research/google-research/tree/master/cluster_gcn