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

Wei-Lin Chiang*
National Taiwan University
r06922166@csie.ntu.edu.tw

Xuanqing Liu*
University of California, Los Angeles
xqliu@cs.ucla.edu

Si Si
Google Research
sisidaisy@google.com

Yang Li
Google Research
liyang@google.com

Samy Bengio
Google Research
bengio@google.com

Cho-Jui Hsieh
University of California, Los Angeles
chohsieh@cs.ucla.edu

Previous GCN Training Algorithms

- in terms of 1) **memory requirement**, 2) **time per epoch** and 3) **convergence speed (loss reduction) per epoch**.

- we denote **N** to be the number of nodes in the graph, **F** the embedding dimension, and **L** the number of layers to analyze classic GCN training algorithms.

Previous GCN Training Algorithms

- Full-batch gradient descent is proposed in the first GCN paper (Semi-Supervised Classification with Graph Convolutional Networks , ICLR 2017)

- O(NFL) memory requirement, the time per epoch is efficient, the convergence of gradient descent is slow since the parameters are updated only once per epoch

[memory: bad; time per epoch: good; convergence: bad]

Previous GCN Training Algorithms

- Mini-batch SGD is proposed in <Inductive Representation Learning on Large Graphs>. NIPS 2017

- mini-batch SGD introduces a signifcant computational overhead due to the neighborhood expansion problem

[memory: good; time per epoch: bad; convergence: good]

Previous GCN Training Algorithms

- VR-GCN proposes to use a variance reduction technique to reduce the size of neighborhood sampling nodes. (Stochastic Training of Graph Convolutional Networks with Variance Reduction, ICML 2018)

- $O(NFL)$ memory requirment,

[memory: bad; time per epoch: good; convergence: good.]

Shortcoming of SGD-based GCN algorithms

- suffer from either a high computational cost that exponentially grows with number of GCN layers

- a large space requirement for keeping the entire graph and the embedding of each node in memory

Cluster-GCN

- We find that the efficiency of a mini-batch algorithm can be characterized by the notion of “embedding utilization”, which is proportional to the number of links between nodes in one batch or within-batch links.

- Memory: O(bFL)

[memory: good; time per epoch: good; convergence: good].

Cluster-GCN

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

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

$$\frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \nabla \text{loss}(y_i, z_i^{(L)}) \qquad (3)$$

Cluster-GCN

$$A = \bar{A} + \Delta = \begin{bmatrix} A_{11} & \cdots & A_{1c} \\ \vdots & \ddots & \vdots \\ A_{c1} & \cdots & A_{cc} \end{bmatrix}$$

and

$$\bar{A} = \begin{bmatrix} A_{11} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & A_{cc} \end{bmatrix}, \Delta = \begin{bmatrix} 0 & \cdots & A_{1c} \\ \vdots & \ddots & \vdots \\ A_{c1} & \cdots & 0 \end{bmatrix},$$

Cluster-GCN

$$A = \bar{A} + \Delta = \begin{bmatrix} A_{11} & \cdots & A_{1c} \\ \vdots & \ddots & \vdots \\ A_{c1} & \cdots & A_{cc} \end{bmatrix}$$

and

$$\bar{A} = \begin{bmatrix} A_{11} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & A_{cc} \end{bmatrix}, \Delta = \begin{bmatrix} 0 & \cdots & A_{1c} \\ \vdots & \ddots & \vdots \\ A_{c1} & \cdots & 0 \end{bmatrix},$$

Cluster-GCN

$$Z^{(L)} = \bar{A}'\sigma(\bar{A}'\sigma(\cdots\sigma(\bar{A}'XW^{(0)})W^{(1)})\cdots)W^{(L-1)} \tag{6}$$

$$= \begin{bmatrix} \bar{A}'_{11}\sigma(\bar{A}'_{11}\sigma(\cdots\sigma(\bar{A}'_{11}X_1W^{(0)})W^{(1)})\cdots)W^{(L-1)} \\ \vdots \\ \bar{A}'_{cc}\sigma(\bar{A}'_{cc}\sigma(\cdots\sigma(\bar{A}'_{cc}X_cW^{(0)})W^{(1)})\cdots)W^{(L-1)} \end{bmatrix}$$

$$\mathcal{L}_{\bar{A}'} = \sum_t \frac{|\mathcal{V}_t|}{N}\mathcal{L}_{\bar{A}'_{tt}} \quad \text{and} \quad \mathcal{L}_{\bar{A}'_{tt}} = \frac{1}{|\mathcal{V}_t|}\sum_{i\in\mathcal{V}_t}\text{loss}(y_i, z_i^{(L)}). \tag{7}$$

Cluster-GCN

Algorithm 1: Cluster GCN

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

Partition graph nodes into c clusters $\mathcal{V}_1, \mathcal{V}_2, \cdots, \mathcal{V}_c$ by METIS;
for $iter = 1, \cdots,$ *max_iter* **do**
    Randomly choose q clusters, $t_1, \cdots, t_q$ from $\mathcal{V}$ without replacement;
    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 $A_{\bar{\mathcal{V}},\bar{\mathcal{V}}}$;
    Compute $g \leftarrow \nabla \mathcal{L}_{A_{\bar{\mathcal{V}},\bar{\mathcal{V}}}}$ (loss on the subgraph $A_{\bar{\mathcal{V}},\bar{\mathcal{V}}}$) ;
    Conduct Adam update using gradient estimator g
Output: $\{W_l\}_{l=1}^{L}$

Issues of training deeper GCNs

$$X^{(l+1)} = \sigma((A' + I)X^{(l)}W^{(l)}) \tag{9}$$

$$\tilde{A} = (D + I)^{-1}(A + I), \tag{10}$$

$$X^{(l+1)} = \sigma((\tilde{A} + \lambda \mathrm{diag}(\tilde{A}))X^{(l)}W^{(l)}). \tag{11}$$