OPT2021: 13th Annual Workshop on Optimization for Machine Learning

Community-based Layerwise Distributed Training of Graph Convolutional Networks

Hongyi Li LIHONGYI@STU.XIDIAN.EDU.CN
The State Key Laboratory of Integrated Service Networks, Xidian University, Xi'an, Shaanxi, China, 710071

Junxiang Wang JUNXIANG.WANG@EMORY.EDU
Department of Computer Science and Informatics, Emory University, Atlanta, Georgia, USA, 30030

Yongchao Wang YCHWANG@MAIL.XIDIAN.EDU.CN
The State Key Laboratory of Integrated Service Networks, Xidian University, Xi'an, Shaanxi, China, 710071

Yue Cheng YUECHENG@GMU.EDU
Department of Computer Science, George Mason University, Fairfax, Virginia, USA, 22030

Liang Zhao LIANG.ZHAO@EMORY.EDU
Department of Computer Science and Informatics, Emory University, Atlanta, Georgia, USA, 30030

Abstract

The Graph Convolutional Network (GCN) has been successfully applied to many graph-based applications. Training a large-scale GCN model, however, is still challenging: Due to the node dependency and layer dependency of the GCN architecture, a huge amount of computational time and memory is required in the training process. In this paper, we propose a parallel and distributed GCN training algorithm based on the Alternating Direction Method of Multipliers (ADMM) to tackle the two challenges simultaneously. We first split GCN layers into independent blocks to achieve layer parallelism. Furthermore, we reduce node dependency by dividing the graph into several dense communities such that each of them can be trained with an agent in parallel. Finally, we provide solutions for all subproblems in the community-based ADMM algorithm. Preliminary results demonstrate that our proposed community-based ADMM training algorithm can lead to more than triple speedup while achieving the best performance compared with state-of-the-art methods.

1. Introduction

Graphs are prevalent structures in various real-world applications including social networks [12], recommender systems [17], and biology and chemistry networks [6], which has attracted much attention from the deep learning community. Graph Convolutional Network (GCN) is one of the leading graph neural network architectures due to its impressive performance on many downstream tasks (e.g. node classification, link prediction, and graph classification) [10]. However, it is challenging to train GCN efficiently due to two difficulties: **1) Node dependency.** The GCN needs to propagate information among nodes through node interactions in the graph. This means that the loss for each node depends on a large number of neighboring nodes. Such dependency becomes more complex as the GCN goes deeper. **2). Layer dependency.** The interactions between nodes are transmitted through layers. Therefore for the backpropagation algorithm, the gradient of node interactions in one layer relies on that in previous layers. Because of node dependency and layer dependency, training a large-scale GCN requires a lot of computational time and memory: node representations in different

layers are required to be updated in sequential, and all of them are required to be stored in the CPU memory.

In order to address these two challenges simultaneously, in this paper, we propose a distributed and parallel GCN training algorithm based on the Alternating Direction Method of Multipliers (ADMM). This is because ADMM has attained great achievements in training deep neural networks in parallel via layer splitting [15]. Specifically, it breaks a series of layers into independent blocks, in order to alleviate layer dependency. Moreover, the complexity of node dependency can be reduced significantly (i.e. from multi-layer level to one-layer level). Apart from layer splitting, we also partition a graph into independent communities: unlike previous works such as Cluster-GCN [4], which remove inter-community connections and thus degrade performance, we maintain node connections, which contain the first-order and second-order neighboring information, and realize parallel training by multiple agents without performance loss. Preliminary experiments on two benchmark datasets demonstrate that our proposed community-based ADMM algorithm leads to more than triple speedup and achieves superior performance compared with state-of-the-art optimizers such as SGD and Adam.

2. Related Work

This section summarizes previous GCN training methods, which consists of minibatch GCN training as well as parallel and distributed GCN training.

Minibatch GCN training. Many sampling-based training methods are proposed to achieve effective minibatch training on GCN, which can be divided into three categories: *1). node-wise sampling-based methods.* This kind of methods sample neighbors for each node in the minibatch locally based on specific probability. Popular models include GraphSAGE [7], VR-GCN [2] and PinSage [17]. *2). Layer-wise sampling-based methods.* They improve node-wise sampling methods by sampling a certain number of globally important nodes together in one sampling step [3, 5, 8, 23]. *3). subgraph-wise sampling-based methods.* They focus on sampling one or more subgraphs at each iteration for mini-batch training. For example, the Cluster-GCN sampled a dense subgraph with graph partitioning algorithm during training and restricts the neighborhood search within the subgraph [4]. GraphSAINT is another example of subgraph minibatch training [19].

Parallel and distributed GCN training. There has been surge research on achieving node parallelism of GCN training by adopting minibatch GCN training methods. Several methods have been proposed to train different mini-batches in parallel on a single machine [18, 20]. Moreover, much effort has been devoted to involving multiple agents to train GCNs, each of which is responsible for a proportion of nodes, to improve time and space efficiency. Famous systems include PyTorch-biggraph [11], Aligraph [22] and DistDGL [21]. In terms of algorithms, distributed Gradient Descent procedure has been proposed to solve the GCN training problem with [13]. The other line of research is layer-wise parallel training. For example, the DGL-GNN algorithm allows different layers to be trained simultaneously by assigning per-layer greedy objectives [16].

3. Problem Formulation

We formulate the GCN training problem in this section. Let $\mathcal{G}(\mathcal{V};\mathcal{E})$ be an undirected and unweighted graph, where $\mathcal{V}$ and $\mathcal{E}$ are sets of nodes and edges, respectively. $N = |\mathcal{V}|$ is the number of nodes.

$A, D \in \mathbb{R}^{n\times n}$ are an adjacency matrix and a degree matrix, respectively. Then the GCN training problem is formulated mathematically as follows:

Problem 1

$$\min_{\mathbf{W},\mathbf{Z}} \quad R(Z_L, Y) \quad s.t. \quad Z_l = f_l(\tilde{A}Z_{l-1}W_l) \quad (l = 1, \cdots, L-1), \quad Z_L = \tilde{A}Z_{L-1}W_L,$$

where $\mathbf{W} = \{W_l\}_{l=1}^{L}, \mathbf{Z} = \{Z_l\}_{l=1}^{L}$, and $\tilde{A} = (D+I)^{-1/2}(A+I)(D+I)^{-1/2}$ is a normalized adjacency matrix. $Z_0 \in \mathbb{R}^{n\times C_0}$ is an input feature matrix, where each row corresponds to an input feature vector of a node, and C_l is the number of hidden units for the l-th layer. $W_l \in \mathbb{R}^{C_{l-1}\times C_l}$ and $Z_l \in \mathbb{R}^{n\times C_l}$ are the weight matrix and the output for the l-th layer, respectively. $Y \in \mathbb{R}^{n\times C_L}$ is the pre-defined label matrix, and C_L is the number of node classes. f_l is a non-linear activation function for the l-th layer (e.g., ReLU). $R(\cdot)$ is a risk function such as the cross-entropy loss. Problem 1 is difficult to solve due to nonlinear constraints $Z_l = f_l(\tilde{A}Z_{l-1}W_l)$. Therefore we relax it to Problem 2 as follows:

Notations	Descriptions
L	Number of layers.
N	Number of nodes.
A	The adjacency matrix of a graph.
D	The degree matrix of a graph.
W_l	The weight matrix for the l-th layer.
$f_l(\cdot)$	The nonlinear activation function for the l-th layer.
Z_l	The output for the l-th layer.
Z_0	The input feature matrix for the neural network.
Y	The predefined label matrix.
$R(Z_L, Y)$	The risk function for the L-th layer.
C_l	The number of neurons for the l-th layer.

Table 1: Important Notations

Problem 2

$$\min_{\mathbf{W},\mathbf{Z}} \quad R(Z_L, Y) + \frac{\nu}{2}\sum_{l=1}^{L-1} \|Z_l - f_l(\tilde{A}Z_{l-1}W_l)\|_F^2 \quad s.t. \quad Z_L = \tilde{A}Z_{L-1}W_L,$$

where $\nu > 0$ is a tuning parameter. Note that when $\nu \longrightarrow \infty$, Problem 2 approaches Problem 1.

Many graph problems such as node classification and link prediction are applied in large-scale scenarios (e.g. social networks), where the adjacency matrix cannot fit in memory. Motivated by Cluster-GCN [4], we divide the graph $\mathcal{G}$ into M communities by METIS[9], where $\mathcal{V} = \bigcup_{m=1}^{M} \mathcal{V}_m$, $\mathcal{V}_m \cap \mathcal{V}_j = \emptyset (1 \le m < j \le M)$. $n_m = |\mathcal{V}_m|$ is the number of nodes in the m-th community. Each community can be fed to an independent agent for distributed training. A community index set neighboring the m-th community is defined as $\mathcal{N}_m = \{i | \exists (u, v) \in \mathcal{E}, u \in \mathcal{V}_m, v \in \mathcal{V}_i, i \neq m\}$. Figure 1 illustrates the partition of communities, where a graph is split into three communities. $\mathcal{N}_1 = \{3\}$ because c, d in community 1 are commected to g in community 3.

We split $\tilde{A}$ according to the partition of $\mathcal{G}$ as follows:

$$\tilde{A} = \begin{bmatrix} \tilde{A}_{1,1} & \cdots & \tilde{A}_{1,M} \\ \vdots & \ddots & \vdots \\ \tilde{A}_{M,1} & \cdots & \tilde{A}_{M,M} \end{bmatrix},$$

where $\tilde{A}_{m,m} \in \mathbb{R}^{n_m \times n_m}$ represents the adjacency matrix of the m-th community, and $\tilde{A}_{m,j}$ defines the topology between the m-th and j-th communities. Accordingly, Z and Y are partitioned as $Z_l = [Z_{l,1}^T, \cdots, Z_{l,M}^T]^T (l = 1, \cdots, L)$ and $Y = [Y_1^T, \cdots, Y_M^T]^T$. Then Problem 2 is equivalently transformed to the following:

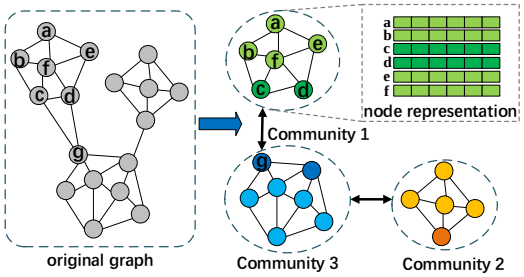


Figure 1: Illustrations of the graph partition: a graph is split into three community.

Problem 3

$$\min_{\mathbf{W},\mathbf{Z}} \sum_{m=1}^{M} R(Z_{L,m}, Y_m) + \frac{\nu}{2}\sum_{l=1}^{L-1}\sum_{m=1}^{M} \|Z_{l,m} - f_l((\tilde{A}_{m,m}Z_{l-1,m} + \sum_{r\in\mathcal{N}_m} \tilde{A}_{m,r}Z_{l-1,r})W_l)\|_F^2$$
$$s.t. \quad Z_{L,m} = (\tilde{A}_{m,m}Z_{L-1,m} + \sum_{r\in\mathcal{N}_m} \tilde{A}_{m,r}Z_{L-1,r})W_L \ (m = 1, \cdots, M).$$

Algorithm 1 The Community-based ADMM Algorithm

Require: Y, Z_0, ν, ρ.
Ensure: $Z_{l,m}, W_l, l = 1, \cdots, L, m = 1, \cdots, M$.
1: **Initialize:** $k = 0$.
2: **while** $W_l^k, Z_{l,m}^k$ not converged **do**
3: Update W_l^{k+1} for different l in parallel.
4: Update $Z_{l,m}^{k+1}$ for different l and m in parallel.
5: Update U_m^{k+1} for different m in parallel.
6: **end while**

4. The community-based ADMM Algorithm

In this section, we propose the ADMM algorithm to solve Problem 3. The augmented Lagrangian is formulated mathematically as follows:

$$\begin{aligned}\mathcal{L}_\rho(\mathbf{W},\mathbf{Z},\mathbf{U}) = &\sum_{m=1}^{M} R(Z_{L,m}, Y_m) + \frac{\nu}{2}\sum_{l=1}^{L-1}\sum_{m=1}^{M} \|Z_{l,m} - f_l((\tilde{A}_{m,m}Z_{l-1,m} + \sum_{r\in\mathcal{N}_m} \tilde{A}_{m,r}Z_{l-1,r})W_l)\|_F^2 \\ &+ \sum_{m=1}^{M} (\langle U_m, (Z_{L,m} - (\tilde{A}_{m,m}Z_{L-1,m} + \sum_{r\in\mathcal{N}_m} \tilde{A}_{m,r}Z_{L-1,r})W_L)\rangle \\ &+ \frac{\rho}{2}\|Z_{L,m} - (\tilde{A}_{m,m}Z_{L-1,m} + \sum_{r\in\mathcal{N}_m} \tilde{A}_{m,r}Z_{L-1,r})W_L\|_F^2), \end{aligned} \tag{1}$$

where $\rho > 0$ is a penalty parameter, and $\mathbf{U} = \{U_m\}_{m=1}^M$ are Lagrangian multipliers. The ADMM algorithm to solve equation 1 is shown in Algorithm 1. Specifically, Lines 3 and 4 update $\mathbf{W}$ (i.e. layerwise training) and $\mathbf{Z}$ (i.e. community-wise training) in parallel, respectively, and Line 7 updates $\mathbf{U}$. All subproblems are discussed in detail as follows. For the sake of simplicity, we define

$$\begin{aligned}\phi(W_l, Z_{l-1}, Z_l) &\triangleq \frac{\nu}{2}\sum_{m=1}^{M} \|Z_{l,m} - f_l((\tilde{A}_{m,m}Z_{l-1,m} + \sum_{r\in\mathcal{N}_m} \tilde{A}_{m,r}Z_{l-1,r})W_l)\|_F^2 \\ &= \frac{\nu}{2}\|Z_l - f_l(\tilde{A}Z_{l-1}W_l)\|_F^2 \ (l = 1, \cdots, L-1),\end{aligned}$$

and

$$\begin{aligned}&\phi(W_L, Z_{L-1}, Z_L, U) \triangleq \sum_{m=1}^{M} (\langle U_m, Z_{L,m} - (\tilde{A}_{m,m}Z_{l-1,m} + \sum_{r\in\mathcal{N}_m} \tilde{A}_{m,r}Z_{l-1,r})W_l\rangle \\ &+ \frac{\rho}{2}\|Z_{L,m} - (\tilde{A}_{m,m}Z_{l-1,m} + \sum_{r\in\mathcal{N}_m} \tilde{A}_{m,r}Z_{l-1,r})W_l\|_F^2) = \langle U, Z_L - \tilde{A}Z_{L-1}W_L\rangle + \frac{\rho}{2}\|Z_L - \tilde{A}Z_{L-1}W_L\|_F^2,\end{aligned}$$

where $U = [U_1^T, \cdots, U_M^T]^T$.

4.1. Update W_l^{k+1}

The variable W_l^{k+1} is updated on agent $M+1$ as follows:

$$W_l^{k+1} \leftarrow \arg\min_{W_l} \mathcal{L}_\rho(\mathbf{W}, \mathbf{Z}^k, \mathbf{U}^k) = \arg\min_{W_l} \begin{cases} \phi(W_l, Z_{l-1}^k, Z_l^k) & l < L \\ \phi(W_L, Z_{L-1}^k, Z_L^k, U^k) & l = L \end{cases}$$

Agent $m (m < M+1)$ needs to send $Z_{l,m}^k (l > 0)$ and U_m^k to agent $M+1$ in advance to form Z_{L-1}^k, Z_L^k, and U^k. Furthermore, solving W_l^{k+1} requires the inverse of $\tilde{A}$, which is usually not inversible. To handle this, we apply the quadratic approximation [14] as follows:

$$W_l^{k+1} \leftarrow \arg\min_{W_l} P_l(W_l; \tau_l^{k+1}), \tag{2}$$

where

$$P_l(W_l; \tau_l^{k+1}) = \begin{cases} \phi(W_l^k, Z_{l-1}^k, Z_l^k) + \langle \nabla_{W_l^k} \phi(W_l^k, Z_{l-1}^k, Z_l^k), W_l - W_l^k \rangle + \frac{\tau_l^{k+1}}{2} \|W_l - W_l^k\|_F^2, l < L \\ \phi(W_L^k, Z_{L-1}^k, Z_L^k, U^k) + \langle \nabla_{W_L^k} \phi(W_L^k, Z_{L-1}^k, Z_L^k, U^k), W_L - W_L^k \rangle + \frac{\tau_l^{k+1}}{2} \|W_L - W_L^k\|_F^2, \end{cases}$$

and $\tau_l^{k+1} > 0$ is a parameter that should satisfy:

$$P_l(W_l^{k+1}; \tau_l^{k+1}) \geq \begin{cases} \phi(W_l^{k+1}, Z_{l-1}^k, Z_l^k), l < L \\ \phi(W_L^{k+1}, Z_{L-1}^k, Z_L^k, U^k), l = L. \end{cases}$$

The solution to equation 2 is:

$$W_l^{k+1} \leftarrow \begin{cases} W_l^k - \nabla_{W_l^k} \phi(W_l^k, Z_{l-1}^k, Z_l^k)/\tau_l^{k+1}, l < L \\ W_L^k - \nabla_{W_L^k} \phi(W_L^k, Z_{L-1}^k, Z_L^k, U^k)/\tau_L^{k+1}, l = L. \end{cases}$$

Obviously, W_l^{k+1} for different layers can be updated in parallel.

4.2. Update $Z_{l,m}^{k+1}$

The update of $Z_{l,m}^{k+1}$ resembles that of W_l^{k+1}. Due to space limit, details are given in Appendix A.

4.3. Update U_m^{k+1}

The variable U_m^{k+1} is updated as follows:

$$U_m^{k+1} \leftarrow U_m^k + \rho(Z_{L,m}^k - (\sum_{r \in \mathcal{N}_m \cup \{m\}} p_{L-1,r\rightarrow m}^k)), \tag{3}$$

where $p_{L-1,r\rightarrow m}^k$ is defined in Appendix A.

5. Experiments

In this section, we evaluate the performance of the proposed community-based ADMM algorithm using two benchmark datasets. Four state-of-the-art optimizers are used as comparison methods in terms of both accuracy and speedup. All experiments were conducted on a 64-bit machine with Intel(R) Xeon(R) Silver 4110 CPU and 64GB RAM. The statistics of two benchmark datasets are shown in Table 2.

5.1. Speedup

In this experiment, we investigate the speedup of the proposed ADMM algorithm on a two-layer GCN model with 1000 hidden units. The activation function was set to the Rectified Linear Unit (ReLU). The loss function was the cross-entropy loss. The running time per epoch was an average of 50 epochs. ρ and ν were both set to 10^{-3} for Amazon Computers and 10^{-4} for Amazon Photo. Specifically, in the Serial ADMM algorithm, we used only one community, and the two layers were trained sequentially; while in the Parallel ADMM algorithm, we divided the original graph into 3 communities that were trained by 3 agents simultaneously, plus applied a layer parallelism scheme.

The training and communication time, as well as speedup, were listed in Table 3 on Amazon Computers and Amazon Photo. The training time on both datasets was reduced by more than 80%. Although the Parallel ADMM involves additional time for communication among agents, it is still around $2\times$ faster than the Serial ADMM method, which demonstrates the effectiveness of the proposed community-based algorithm.

5.2. Accuracy

To validate the accuracy of the proposed community-based ADMM algorithms, we used the same GCN architecture and parameter settings for Serial ADMM and Parallel ADMM algorithms as those in Section 5.1. SGD and its variants are state-of-the-art optimizers for GCN training and hence we used four of them as comparison methods, namely, Adaptive momentum estimation (Adam), Adaptive gradient algorithm (Adagrad), Gradient Descent (GD), and Adaptive learning rate method (Adadelta). For comparison methods, we used the following learning rate for Amazon Computers and Amazon Photo: 10^{-3} (Adam, Adagrad, and Adadelta) and 10^{-1} (GD) based on the optimal training performance.

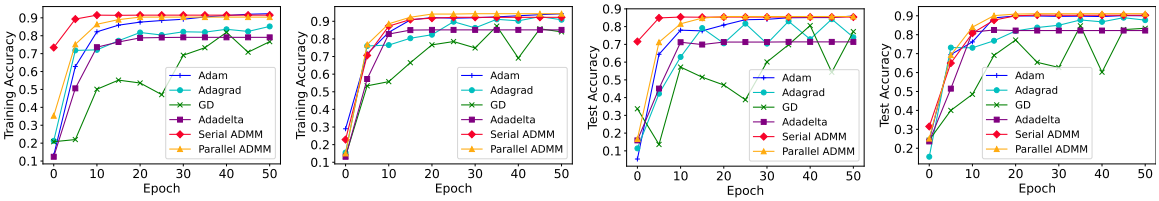
In this section, the accuracy of the proposed serial ADMM and parallel ADMM algorithms is analyzed against all comparison methods. Figure 2 illustrates the training and test accuracy for all training methods on both datasets. The proposed Serial and Parallel ADMM algorithms reach the highest accuracy and outperform most comparison methods except for Adam, which perform almost the same compared to the proposed algorithms when epoch=50. Furthermore, the proposed two ADMM algorithms converge the fastest among all methods, and the convergence speed of Serial ADMM is ahead of that of Parallel ADMM in most situations.

Dataset	Node#	Training Sample#	Test Sample#	Class#	Feature#
Amazon Computers	13752	1000	1000	10	767
Amazon Photo	7650	800	1000	8	745

Table 2: Two benchmark datasets.

	Serial ADMM (sec)	Parallel ADMM (sec)			
Dataset	Total	Training	Communication	Total	Speedup
Amazon Computers	80.82	14.94	9.54	24.48	3.30
Amazon Photo	50.81	8.80	8.27	17.07	2.98

Table 3: Comparison of training and communication time on two datasets.



(a). Training accuracy for Amazon Computers. (b). Training accuracy for Amazon Photo. (c). Test accuracy for Amazon Computers. (d). Test accuracy for Amazon Photo.

Figure 2: Training accuracy and test accuracy of all methods: Serial ADMM algorithm and Parallel ADMM algorithm outperform most of comparison methods in two datasets.

6. Conclusion

In this paper, we present the community-based Alternating Direction Methods of Multipliers (ADMM) algorithm to achieve both node parallelism and layer parallelism on training large-scale Graph Convolutional Networks (GCNs). Preliminary experimental results on benchmark datasets show that the community-based ADMM method leads to huge speedup and achieves excellent performance compared to state-of-the-art optimizers.

References

[1] Amir Beck and Marc Teboulle. A Fast Iterative Shrinkage-Thresholding Algorithm. *Society for Industrial and Applied Mathematics Journal on Imaging Sciences*, 2(1):183–202, 2009. ISSN 1936-4954.

[2] 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, 10–15 Jul 2018.

[3] Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. In *Sixth International Conference on Learning Representations*, 2018.

[4] 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*, pages 257–266, 2019.

[5] Tyler Derr, Yao Ma, Wenqi Fan, Xiaorui Liu, Charu Aggarwal, and Jiliang Tang. Epidemic graph convolutional network. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 160–168, 2020.

[6] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alan Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[7] 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*, pages 1025–1035, 2017.

[8] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. Adaptive sampling towards fast graph representation learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[9] Karypis, George, Kumar, and Vipin. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 1998.

[10] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[11] Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. Pytorch-biggraph: A large-scale graph embedding system. In *Systems for ML*, 2019.

[12] Jiezhong Qiu, Jian Tang, Hao Ma, Yuxiao Dong, Kuansan Wang, and Jie Tang. Deepinf: Social influence prediction with deep learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2110–2119, 2018.

[13] Simone Scardapane, Indro Spinelli, and Paolo Di Lorenzo. Distributed training of graph convolutional networks. *IEEE Transactions on Signal and Information Processing over Networks*, 7: 87–100, 2021. doi: 10.1109/TSIPN.2020.3046237.

[14] Junxiang Wang, Fuxun Yu, Xiang Chen, and Liang Zhao. ADMM for efficient deep learning with global convergence. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 111–119, 2019. ISBN 9781450362016. doi: 10.1145/3292500.3330936.

[15] Junxiang Wang, Zheng Chai, Yue Cheng, and Liang Zhao. Toward model parallelism for deep neural network based on gradient-free ADMM framework. In *20th IEEE International Conference on Data Mining*, Virtual Event, Sorrento, Italy, 2020.

[16] Yewen Wang, Jian Tang, Yizhou Sun, and Guy Wolf. Decoupled greedy learning of graph neural networks. *OPT2020: 12th Annual Workshop on Optimization for Machine Learning*, 2020.

[17] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 974–983, 2018.

[18] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Accurate, efficient and scalable graph embedding. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 462–471. IEEE, 2019.

[19] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. In *ICLR*, 2020.

[20] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Accurate, efficient and scalable training of graph neural networks. *Journal of Parallel and Distributed Computing*, 147:166–183, 2021. ISSN 0743-7315.

[21] Da Zheng, Chao Ma, Minjie Wang, Jinjing Zhou, Qidong Su, Xiang Song, Quan Gan, Zheng Zhang, and George Karypis. Distdgl: distributed graph neural network training for billion-scale graphs. In *2020 IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms (IA3)*, pages 36–44. IEEE, 2020.

[22] Rong Zhu, Kun Zhao, Hongxia Yang, Wei Lin, Chang Zhou, Baole Ai, Yong Li, and Jingren Zhou. Aligraph: a comprehensive graph neural network platform. *Workshop on AI Systems at SOSP 2019 - System for ML*, 2019.

[23] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. Layer-dependent importance sampling for training deep and large graph convolutional networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Appendix A. Update $Z_{l,m}^{k+1}$

The variable $Z_{l,m}^{k+1}$ is updated as follows:

$$Z_{l,m}^{k+1} \leftarrow \text{argmin}_{Z_{l,m}} \mathcal{L}_\rho(\mathbf{W}^{k+1}, \mathbf{Z}, \mathbf{U}^k).$$

Notably, as output variables for intermediate layers (i.e. Z_l) are involved in two constraints in Problem 1, updating $Z_{l,m}^{k+1}(l = 1, \cdots, L-1)$ requires $\{\tilde{A}_{m,r} Z_{l,r}^k W_{l+1}^{k+1}\}$, $\{\tilde{A}_{r,r'} Z_{l,r'}^k W_{l+1}^{k+1}\}$, $\{Z_{l+1,r}^k\}$, and $\{\tilde{A}_{m,r} Z_{l-1,r}^k W_l^{k+1}\}$, where $r \in \mathcal{N}_m \cup \{m\}$ and $r' \in \mathcal{N}_r \cup \mathcal{N}_m \backslash \{m\}$. In other words, the update of $Z_{l,m}^{k+1}$ requires information of second-order neighbors, which suffers from the bottleneck of the inherent neighbor explosion. To tackle this problem, the information from second-order neighbors can be conveyed via first-order neighbors, which are detailed as follows.

The first-order information is defined by:

$$p_{l,r\to m}^k \triangleq \tilde{A}_{m,r} Z_{l,r}^k W_{l+1}^{k+1} (l = 0, \cdots, L-1).$$

The second-order information is defined in the following:

$$\begin{aligned} s_{l,r\to m}^k = [s_{l,r\to m}^{k,1}, s_{l,r\to m}^{k,2}] &\triangleq \begin{cases} [Z_{l+1,r}^k, \sum_{r' \in \mathcal{N}_r \cup \{r\} \backslash \{m\}} \tilde{A}_{r,r'} Z_{l,r'}^k W_{l+1}^{k+1}] \\ [Z_{L,r}^k - \sum_{r' \in \mathcal{N}_r \cup \{r\} \backslash \{m\}} \tilde{A}_{r,r'} Z_{L-1,r'}^k W_L^{k+1}, U_r^k] \end{cases} \\ &= \begin{cases} [Z_{l+1,r}^k, \sum_{r' \in \mathcal{N}_r \cup \{r\} \backslash \{m\}} p_{l,r'\to r}^k] & l = 0, \cdots, L-2 \\ [Z_{L,r}^k - \sum_{r' \in \mathcal{N}_r \cup \{r\} \backslash \{m\}} p_{L-1,r'\to r}^k, U_r^k] & l = L-1. \end{cases} \end{aligned} \tag{4}$$

We can see from equation 4 that the second-order information forwarded by r to m can easily be constructed by community r through aggregating its received first-order information $p_{l,r'\to r}^k$ from all $r' \in \mathcal{N}_r \cup \{r\} \backslash \{m\}$. We further define $\mathbf{p}_{l,m}^k \triangleq \{p_{l,r\to m}^k | r \in \mathcal{N}_m\}$ and $\mathbf{s}_{l,m}^k \triangleq \{s_{l,r\to m}^k | r \in \mathcal{N}_m\}$. Then the objective for $Z_{l,m}^{k+1}$ can be modified as:

$$\begin{aligned} Z_{l,m}^{k+1} \leftarrow \text{argmin}_{Z_{l,m}} & \frac{\nu}{2} \|Z_{l,m} - f_l(\sum_{r \in \mathcal{N}_m \cup \{m\}} p_{l-1,r\to m}^k)\|_F^2 \\ & + \frac{\nu}{2} \|Z_{l+1,m}^k - f_{l+1}(\tilde{A}_{m,m} Z_{l,m} W_{l+1}^{k+1} + \sum_{r \in \mathcal{N}_m} p_{l,r\to m}^k)\|_F^2 + \sum_{r \in \mathcal{N}_m} \frac{\nu}{2} \|s_{l,r\to m}^{k,1} - f_{l+1}(\tilde{A}_{r,m} Z_{l,m} W_{l+1}^{k+1} + s_{l,r\to m}^{k,2})\|_F^2 \\ & \triangleq \psi(Z_{l,m}, Z_{l+1,m}^k, W_{l+1}^{k+1}, \mathbf{p}_{l,m}^k, \mathbf{p}_{l-1,m}^k, \mathbf{s}_{l,m}^k) (l = 1, \cdots, L-2), \end{aligned} \tag{5}$$

$$\begin{aligned} Z_{L-1,m}^{k+1} \leftarrow \arg\min_{Z_{L-1,m}} & \frac{\nu}{2} \|Z_{L-1,m} - f_{L-1}(\sum_{r \in \mathcal{N}_m \cup \{m\}} p_{L-2,r\to m}^k)\|_F^2 \\ & + \langle U_m^k, Z_{L,m}^k - (\tilde{A}_{m,m} Z_{L-1,m} W_L^{k+1} + \sum_{r \in \mathcal{N}_m} p_{L-1,r\to m}^k) \rangle + \frac{\rho}{2} \|Z_{L,m}^k - (\tilde{A}_{m,m} Z_{L-1,m} W_L^{k+1} + \sum_{r \in \mathcal{N}_m} p_{L-1,r\to m}^k)\|_F^2 \\ & + \sum_{r \in \mathcal{N}_m} (\langle s_{L-1,r\to m}^{k,2}, s_{L-1,r\to m}^{k,1} - \tilde{A}_{r,m} Z_{L-1,m} W_L^{k+1} \rangle + \frac{\rho}{2} \|s_{L-1,r\to m}^{k,1} - \tilde{A}_{r,m} Z_{L-1,m} W_L^{k+1}\|_F^2) \\ & \triangleq \psi(Z_{L-1,m}, Z_{L,m}^k, W_L^{k+1}, \mathbf{p}_{L-1,m}^k, \mathbf{p}_{L-2,m}^k, \mathbf{s}_{L-1,m}^k), \end{aligned} \tag{6}$$

and

$$\begin{aligned} Z_{L,m}^{k+1} \leftarrow \arg\min_{Z_{L,m}} R(Z_{L,m}, Y_m) + \langle U_m^k, Z_{L,m} - (\tilde{A}_{m,m} Z_{L-1,m}^k W_L^{k+1} + \sum_{r \in \mathcal{N}_m} p_{L-1,r\to m}^k) \rangle \\ + \frac{\rho}{2} \| Z_{L,m} - (\tilde{A}_{m,m} Z_{L-1,m}^k W_L^{k+1} + \sum_{r \in \mathcal{N}_m} p_{L-1,r\to m}^k) \|_F^2 \\ \triangleq \psi(Z_{L,m}, Z_{L-1,m}^k, W_L^{k+1}, \mathbf{p}_{L-1,m}^k, U_m^k). \end{aligned} \tag{7}$$

Obviously, $Z_{l,m}^{k+1}$ for different m and l can all be updated in parallel. Furthermore, community m should receive $p_{l,r\to m}^k (l < L)$ and $s_{l,r\to m}^k (l < L)$ from all its neighbor communities r before updating $Z_{l,m}^{k+1}$. In addition, the close-form solution to $Z_{l,m}^{k+1} (l < L)$ requires time-consuming matrix inverse operation. Similar to update W, the quadratic approximation technique is applied as follows:

$$Z_{l,m}^{k+1} \leftarrow \arg\min_{Z_{l,m}} Q_l(Z_{l,m}; \theta_{l,m}^{k+1}), \tag{8}$$

where

$$\begin{aligned} Q_{l,m}(Z_{l,m}; \theta_{l,m}^{k+1}) \triangleq \psi(Z_{l,m}^k, Z_{l+1,m}^k, W_{l+1}^{k+1}, \mathbf{p}_{l,m}^k, \mathbf{p}_{l-1,m}^k, \mathbf{s}_{l,m}^k) + \frac{\theta_{l,m}^{k+1}}{2} \| Z_{l,m} - Z_{l,m}^k \|_F^2 \\ + \langle \nabla_{Z_l^k} \psi(Z_{l,m}^k, Z_{l+1,m}^k, W_{l+1}^{k+1}, \mathbf{p}_{l,m}^k, \mathbf{p}_{l-1,m}^k, \mathbf{s}_{l,m}^k), Z_{l,m} - Z_{l,m}^k \rangle, l < L, \end{aligned}$$

and $\theta_{l,m}^{k+1} > 0$ is a parameter that should satisfy:

$$Q_{l,m}(Z_{l,m}^{k+1}; \theta_{l,m}^{k+1}) \geq \psi(Z_{l,m}^{k+1}, Z_{l+1,m}^k, W_{l+1}^{k+1}, \mathbf{p}_{l,m}^k, \mathbf{p}_{l-1,m}^k, \mathbf{s}_{l,m}^k), l < L \tag{9}$$

The solution is:

$$Z_{l,m}^{k+1} \leftarrow Z_{l,m}^k - \nabla_{Z_{l,m}^k} \psi(Z_{l,m}^k, Z_{l+1,m}^k, W_{l+1}^{k+1}, \mathbf{p}_{l,m}^k, \mathbf{p}_{l-1,m}^k, \mathbf{s}_{l,m}^k) / \theta_{l,m}^{k+1}, l < L. \tag{10}$$

Finally, equation 7 (i.e. $l = L$) can be solved directly via Fast Iterative Soft-Thresholding Algorithm (FISTA) [1].