ADVANCING GRAPHSAGE WITH A DATA-DRIVEN NODE SAMPLING

Jihun Oh
Samsung Research
Samsung Electronics, co., Seoul, Republic of Korea
jihun2331.oh@samsung.com

Kyunghyun Cho
Department of Computer Sicence
New York University, New York, United States
kyunghyun.cho@nyu.edu

Joan Bruna
Department of Computer Sicence
New York University, New York, United States
bruna@cims.nyu.edu

ABSTRACT

As an efficient and scalable graph neural network, GraphSAGE has enabled an inductive capability for inferring unseen nodes or graphs by aggregating subsampled local neighborhoods and by learning in a mini-batch gradient descent fashion. The neighborhood sampling used in GraphSAGE is effective in order to improve computing and memory efficiency when inferring a batch of target nodes with diverse degrees in parallel. Despite this advantage, the default uniform sampling suffers from high variance in training and inference, leading to sub-optimum accuracy.

We propose a new data-driven sampling approach to reason about the real-valued importance of a neighborhood by a non-linear regressor, and to use the value as a criterion for subsampling neighborhoods. The regressor is learned using a value-based reinforcement learning. The implied importance for each combination of vertex and neighborhood is inductively extracted from the negative classification loss output of GraphSAGE. As a result, in an inductive node classification benchmark using three datasets, our method enhanced the baseline using the uniform sampling, outperforming recent variants of a graph neural network in accuracy.

1 INTRODUCTION

Machine learning on graph-structured network data has proliferated in a number of important applications. To name a few, it shows great potential in a chemical prediction problem (Gilmer et al. (2017)), a protein functions understanding, and particle physics experiments (Henrion et al. (2017); Choma et al. (2018)). Learning the representation of structural information about a graph discovers a mapping that embeds nodes (or sub-graphs), as points in a low-dimensional vector space. Graph neural network algorithms based on neighborhood aggregation, addressed the problem by leveraging a node's attributes (Kipf & Welling (2016); Hamilton et al. (2017); Pham et al. (2017)). The GraphSAGE algorithm (Hamilton et al. (2017)) recursively subsamples by uniform sampling a fixed number of nodes from local neighborhoods over multiple hops, and learns a set of aggregator models that aggregate the hidden features of the subsampled nodes by backtracking toward the origin. The sampling approach keeps the computational footprint of each batch in parallel computing fixed. However, despite the comprehensive features of GraphSAGE, unbiased random sampling with uniform distribution causes high variance in training and testing, which leads to suboptimal accuracy. In the present work, we propose a novel method to replace the subsampling algorithm in GraphSAGE with a data-driven sampling algorithm, trained with Reinforcement Learning.

2 PRELIMINARIES: GRAPHSAGE

GraphSAGE (Hamilton et al. (2017)) performs local neighborhood sampling and then aggregation of generating the embeddings of the sampled nodes. The sampling step provides the benefits such

that the computational and memory complexity is constant with respect to the size of a graph. Once the target node, $v \in V$, is determined, a fixed set of neighborhoods, u^k, is sampled as follows:

$$\begin{aligned} &u^0 = \{v\}, \\ &u^k = \cup_{\nu \in u^{k-1}} \mathcal{S}(A_\nu, N^k), \quad k = 1, 2, ..., K, \end{aligned} \tag{1}$$

where A_ν is a set of neighboring nodes of ν and N^k is the sample size at depth k. $\mathcal{S}(A_\nu, N^k)$ is a sampler from a uniform distribution $U(1, deg(v))$ as a default setting. This way the receptive field of a single node grows with respect to the number of layers, K, so the size of $\cup_{k=1}^{K} u^k$ is $\prod_{k=1}^{K} N^k$. After the sampling, we aggregate the embeddings of nodes in the sampled set toward the original node v.

The initial node embeddings, h_u^0 for a sampled set u, are the input node attributes (features) x_v with the dimension of M:

$$h_u^0 = x_v, \quad \forall v \in \{v\} \cup u^1 \cup \cdots \cup u^K. \tag{2}$$

The mean_concat aggregator averages the embeddings, $h_{\nu \in \mathcal{N}(u)}^{k-1}$, of the neighboring nodes, $\mathcal{N}(u)$, of a set of sampled node u. Then, that aggregated neighbor embedding is combined by concatenation with the embedding h_u^{k-1} of a node u to assign a new embedding h_u^k into the node. If the concatenation is changed into the addition, it becomes the mean_add aggregator.

$$\begin{aligned} &\text{for} \quad k = 1, 2, ..., K \quad \text{do} \\ &\quad \text{for} \quad u \in \{v\} \cup u^1 \cup \cdots \cup u^{K-k} \quad \text{do} \\ &\qquad h_u^k = \sigma\left\{ \left(W_\nu^k \sum_{\nu \in \mathcal{N}(u)} \frac{h_\nu^{k-1}}{|\nu|} \right) \| \left(W_u^k h_u^{k-1} \right) \right\}, \end{aligned} \tag{3}$$

where W_ν^k and W_u^k with a size of $M' \times M$ at the first layer and $M' \times M'$ at the remaining layers are weight matrices that are shared among nodes in the network layer k. M' is the hidden feature dimension, and $\sigma(\cdot)$ is a non-linear function, such as a rectified linear unit, defined as $\max(0, x)$. The operator $||$ indicates the concatenation of two vectors. Afterward, the new embedding, h_u^k, is normalized. After finishing K-layer processing, the final embedding vector, h_u^K, is generated. This goes to a classifying layer to predict C-classes. The GraphSAGE model is trained to minimize classification cross-entropy loss.

$$L\left(\hat{y}, y\right) = -\sum_{v \in V} \sum_{i=1}^{C} y_i \log \hat{y}_i, \quad \forall y \in Y \tag{4}$$

3 METHOD

3.1 VALUE FUNCTION-BASED REINFORCEMENT LEARNING FOR NODE SAMPLING

To replace the previous uniform sampler, we consider a Reinforcement Learning approach which helps learning how to quickly find a good sampling distribution in a new dataset. A per-step reward, R^k, is a negative value of cross-entropy loss computed at the node v given a k-hop uniformly subsampled neighborhood as well as a directly connected 1-hop neighborhood, u^1. Note that the per-step reward is a batch-wise value not applying summation over a mini-batch of target nodes, $v \in V$:

$$R_{v,u}^k = \sum_{i=1}^{C} y_{v,i} \log \hat{y}_{v,i}^k = \sum_{i=1}^{C} y_{v,i} \log \mathcal{F}_\theta(v | u^1 \cup \cdots \cup u^k), \quad u \in u^1, \tag{5}$$

where $\mathcal{F}_\theta$ is the aggregator of GraphSAGE and inputs a target node v and k-hop subsampled neighborhood, $u^1 \cup \cdots \cup u^k$. A per-step visit count, $C_{v,u}^k$, records how many times (v, u) is indexed.

$$C_{v,u}^k \leftarrow C_{v,u}^k + 1, \quad u \in u^1 \tag{6}$$

The layer depth of aggregator is equal to the number of hop (K), as seen in the iteration count of the outer loop surrounding the aggregator (equation 3). To produce per-step rewards, GraphSAGE predicts the classes, $\hat{y}^k$, at all the intermediate layers. To do so, we add the auxiliary classifying layers at every intermediate layer beside the final layer. We consider a return G consisting of the discounted sum of per-step rewards propagated from the first hop to the final K-th hop:

$$G_{v,u} = R_{v,u}^1 + \gamma R_{v,u}^2 + ... + \gamma^{K-1} R_{v,u}^K = \sum_{k=0}^{K-1} \gamma^k R_{v,u}^{k+1}, \tag{7}$$

where $\gamma \in (0, 1]$ is a discount factor that discounts the contribution from the future reward. In other words, with a lower γ, we impose that a neighborhood at a closer distance has more influence on the return $G_{v,u}$. In order to avoid the overhead computing all the per-step rewards, we explore an approximation scheme where we set R^k to zero if $k < K$. Equation 7 can be replaced with a version of last-hop learning approximating all-hop learning; $G_{v,u} = R^K_{v,u}$. A visit count, $C_{v,u}$, sums all the per-step visit counts.

$$C_{v,u} = \sum_{k=1}^{K} C^k_{v,u} \tag{8}$$

This return is optimized with respect to a policy π using Reinforcement Learning. The inputs of the policy are a target node and candidates of its neighborhood, and the output action space is either 1 or 0 , indicating being selected as a subsample or not. The value function associated to this policy is denoted by $\mathcal{V}_{v,u}$; we recall that it is the expected return, obtained by division of $G_{v,u}$ by $C_{v,u}$, under the policy starting from a target node v to a neighboring node u. The relationship between the value function and the neighboring node $u \in u^1$ connected to the target node v is defined as follows:

$$\mathcal{V}_{v,u} = \mathbb{E}_\pi \left[G | S = (v, u) \right] = \frac{G_{v,u}}{C_{v,u}}, \quad \mathcal{V} \in \mathbb{R}^{|V| \times \max(\deg)}, \quad u \in u^1. \tag{9}$$

3.2 Nonlinear Regressor to Model the Value Function

A possible state (v, u) is not confined to a finite set of nodes observed in training. That is because it is assumed the graph is evolving; that is, unseen nodes can be observed during testing. Thus, We consider a function approximation to the value function $\mathcal{V}$ using non-linear combination of attributes at state (v, u).

$$\hat{\mathcal{V}}_{v,u} = \mathcal{G}_\theta(v, u) = -\exp\left(\sigma\left(W\left(x_v || x_u\right) + b\right)\right), \quad x_v, x_u \in \mathbb{R}^M, \quad u \in \mathcal{N}(v) \tag{10}$$

where let x_v and x_u be M-dimensional input vectors (attributes) of a node v and each member of a neighborhood, $u \in \mathcal{N}(v)$, respectively. θ denotes the weights of a differentiable non-linear regressor function, $\mathcal{G}$. A weight matrix W with a size of $1 \times 2M$ and bias b are the parameters of a single perceptron layer to be learned. This model is trained to minimize the $l2$-norm between the true value function, $\mathcal{V}_{v,u}$, obtained in equation 9 and the output, $\hat{\mathcal{V}}_{v,u}$, using mini-batch gradient descent optimization. The learned weights are shared in sampling neighborhood at all depths.

3.3 Node Sampling and Acceleration

For subsampling a set of neighborhood u^k of a set of node u^{k-1} by reinforcement learning, we redefine the neighborhood sampling function, S in equation 1, to include the non-linear regressor trained in subsection 3.2.

$$u^k = \cup_{\nu \in u^{k-1}} \mathcal{S}(\mathcal{G}_\theta(\nu, A_\nu), N^k), \tag{11}$$

where A_ν is a set of neighboring nodes of $\nu \in u^{k-1}$. $\mathcal{G}_\theta$ is the non-linear regressor. N^k is the subsample size at the k-th hop. Based on the estimated value functions over the neighborhood, sorting the neighboring nodes in descending order and selecting top N^k decrease the computational efficiency. To alleviate complexity and obtain the benefits of parallelism, all immediate neighbors are partitioned into $\mathcal{B} = N^k$ groups. Then, the argmax operation is executed in parallel to find the neighbor with the maximal predicted return in each batch. This scheme reduces the complexity to $O(n)$ in sequential mode or $O(n/\mathcal{B})$ in parallel mode:

$$\mathcal{S}(\mathcal{G}_\theta(\nu, A_\nu), N^k) = \{\mathrm{arg}_{i \in A_{\mathcal{B}(\nu)}} \max(\mathcal{G}_\theta(\nu, A_{\mathcal{B}(\nu)}, N^k))\}, \tag{12}$$

where let $\mathcal{B}(\nu)$ be N^k groups of evenly partitioned neighborhood.

4 Experiments

4.1 Experimental Setup

For a supervised classification task on a large-scale graph in an inductive setting, we used protein-protein interaction (PPI) (Zitnik & Leskovec (2017)), Reddit (Grover & Leskovec (2016)), and PubMed (Kipf & Welling (2016)) datasets. The classification accuracy metric is a micro F1 score,

Table 1: Reinforcement learning based sampling using uniform (baseline) vs. all-hop rewards vs. first-hop reward vs. last-hop reward; for parameter settings, a hidden dimension is 512, sample size is 30 for all layers, and discount rate γ is 0.9; two or three mean_concat aggregator layers plus one classification layer are constructed. Training ran for ten epochs with a batch size of 32. The shown Micro F1 score is averaged for five runs. Here, the first-hop RL is by using a very small $\gamma = 0.001$.

	Uniform	All-hop RL	First-hop RL	Last-hop RL
		PPI		
Two-layer (K=2)	0.674	**0.755**	0.743	0.742
Three-layer (K=3)	0.780	**0.846**	0.844	0.843
		Reddit		
Two-layer (K=2)	0.950	**0.954**	0.953	0.952
Three-layer (K=3)	0.959	**0.963**	0.961	0.961
		PubMed		
Two-layer (K=2)	0.879	0.881	0.882	**0.885**
Three-layer (K=3)	0.877	0.888	0.888	**0.889**

combining a recall and a precision, that is commonly used in the benchmark task. We tested on the mean_concat aggregator in equation 3 with 2 or 3 layers (K). The default hidden feature dimension size M' is 512 in all hidden layers. The neighborhood sample size is set to 30 at all hops. We use the Adam optimizer (Kingma & Ba (2014)) and ran 10 epochs with a batch size of 32 and a learning rate of 0.01. When optimizing the non-linear regressor, we ran 50 epochs with a batch size of 512 and a learning rate of 0.001.

All the models were implemented in Python 2.7 and Tensorflow 1.12.0. Our computing environment was a single Tesla P40 GPU, 24GB memory on GPU with CUDA 9.2 and cuDNN 7.2.1 in the CentOS Linux 7. Our code can be downloaded from `https://github.com/oj9040/GraphSAGE_RL.git`.

4.2 RESULTS

In Table 1, the RL-based training showed over the baseline method relative improvement of 12.0% (two-layer) and 8.5% (three-layer) for the PPI dataset. The all-hop reward training exhibited slight superiority over last-hop reward, but the difference was not as large as the difference over the baseline. It supports the use of the last-hop approximation which is computationally more efficient. The effect of RL-based sampling was shown differently according to the distribution type and range of the observed value function. It was close to the Gaussian distribution spanned over a high and wide range for the PPI dataset while it was closely characterized by the Rayleigh distribution concentrated on a very low and narrow range for the Reddit or PubMed dataset. We can infer from the high concentration on near-zero values that the graph nodes are distributed over the relatively regular space. This may cause a marginal advantage of the RL-based sampling over the uniform sampling.

In Table 2, GraphSAGE with the RL-based sampling (*) achieved the runner-up accuracy on the PPI and the best on the Reddit and PubMed datasets. Training for longer epochs helped improving the accuracy (#3 vs. #5, #4 vs. #6). Beside the default GraphSAGE of mean_concat aggregator and a sample size of [30, 30], a better compute-optimized network consisting of a mean_add aggregator and a smaller and hop-wise decreasing sample size of [25, 10] (suggested by Hamilton et al. (2017)) was also performed (#7, #8). The parameter size of the mean_add aggregator was approximately two third smaller than mean_concat (refer to Par (MB)). Nevertheless, the accuracy of mean_add was similar to or higher than mean_concat when our proposed sampling method was applied (#6 vs. #8). The proposed method is proven to be practical and useful among these cutting-edge methods from the perspectives of high-ranked accuracy and memory and computing efficiency.

Table 2: A summary of comparisons with cutting-edge methods, such as FastGCN (Chen et al. (2018)), graph attention network (Velickovic et al. (2017)), and GraphSAGE (GS) with a mean_concat (A1) aggregator and a sample size of [30, 30], including our proposed sampling method (*), all of which are two-layer networks. A better compute-optimized version (bold), consisting of mean_add (A2) aggregator and the sample size of [25, 10], is also performed. Testing time was measured in seconds for all test nodes. The shown Micro F1 score is averaged for five runs. The results from default settings, #3 and #4, are referred from Table 1. The 'oom' indicates the runtime error due to out-of-memory.

		PPI			Reddit			PubMed		
#	Method	F1	Time (s)	Par (MB)	F1	Time (s)	Par (MB)	F1	Time (s)	Par (MB)
1	FastGCN_100ep	0.730	0.10	1.4	0.945	0.53	2.4	0.876	0.040	0.037
2	GAT_100ep	0.973	1.29	12.0		oom		0.863	0.243	6.2
3	GS_A1_[30,30]_10ep	0.674	0.18	4.7	0.950	5.21	6.6	0.879	0.079	6.0
4	*GS_A1_[30,30]_10ep	0.755	0.29	4.7	0.954	14.73	6.6	0.881	0.210	6.0
5	GS_A1_[30,30]_100ep	0.746	0.18	4.7	0.950	5.21	6.6	0.888	0.079	6.0
6	*GS_A1_[30,30]_100ep	0.785	0.29	4.7	0.955	14.73	6.6	0.890	0.210	6.0
7	GS_A2_[25,10]_100ep	0.713	0.15	2.5	0.942	2.84	4.5	0.872	0.023	4.0
8	***GS_A2_[25,10]_100ep**	**0.813**	**0.24**	**2.5**	**0.954**	**6.99**	**4.5**	**0.898**	**0.097**	**4.0**

5 CONCLUSION

We introduced a novel data-driven neighborhood sampling approach, learned by a Reinforcement Learning, replacing random sampling with uniform distribution in GraphSAGE (Hamilton et al. (2017)). In order to embed nodes in a large-scale graph using limited computing and memory resources, it is crucial to sample a small set of neighboring nodes with high importance. For the supervised classification task in an inductive setting, we empirically showed that the proposed sampling method improves the node classification accuracy over the uniform sampling based GraphSAGE.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous referees for their valuable comments and helpful suggestions. The authors collaborated with the Center for Data Science, New York University, New York, NY, USA, and were funded by Samsung Research, Samsung Electronics Co., Seoul, Republic of Korea. We express special thanks to Dr. Daehyun Kim, Dr. Myungsun Kim, and Yongwoo Lee at Samsung Research for their substantial help in supporting this collaboration.

REFERENCES

Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: Fast Learning With Graph Convolu- Tional Networks Via Importance Sampling. *Iclr*, pp. 1–15, 2018. URL https://openreview.net/pdf?id=rytstxWAW.

Nicholas Choma, Federico Monti, Lisa Gerhardt, Tomasz Palczewski, Zahra Ronaghi, Prabhat Prabhat, Wahid Bhimji, Michael Bronstein, Spencer Klein, and Joan Bruna. Graph neural networks for icecube signal classification. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 386–391. IEEE, 2018.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural Message Passing for Quantum Chemistry. 2017. ISSN 0022-2623. doi: 10.1021/acs.jmedchem.7b01484. URL http://arxiv.org/abs/1704.01212.

Aditya Grover and Jure Leskovec. Node2Vec. *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discov. Data Min. - KDD '16*, pp. 855–864, 2016. ISSN 2154-817X. doi: 10.1145/2939672.2939754. URL http://dl.acm.org/citation.cfm?doid=2939672.2939754.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pp. 1024–1034, 2017.