2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing Workshops (CCGridW) | 979-8-3503-0208-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/CCGridW59191.2023.00076

Performance Modelling of Graph Neural Networks

Pranjal Naman † and Yogesh Simmhan
Department of Computational and Data Sciences, Indian Institute of Science, Bangalore 560012 INDIA
Email: pranjalnaman@iisc.ac.in, simmhan@iisc.ac.in

***Abstract*—Recent years have witnessed a rapid rise in the popularity of Graph Neural Networks (GNNs) that address a wide variety of domains using different architectures. However, as relevant graph datasets become diverse in size, sparsity and features, it becomes important to quantify the effect of different graph properties on the training time for different GNN architectures. This will allow us to design compute-aware GNN architectures for specific problems, and further extend this for distributed training. In this paper, we formulate the calculation of the Floating Point Operations (FLOPs) required for a single forward pass through layers of a GNN. We report the analytical calculations for GraphConv and GraphSAGE models and compare against their profiling results for 10 graphs with varying properties. We observe that there is a strong correlation between our theoretical expectation of the number of FLOPs and the experimental execution time for a forward pass.**

I. Introduction

Graph Neural Networks (GNNs) have received remarkable attention recently. They have been used for a variety of tasks ranging from node classification [1], graph classification [2] to edge predictions [1], with applications to various fields like financial fraud detection [3], recommender systems [4], etc. GNNs aim to map the high-dimensional node or edge features to low-dimensional embeddings using either a *spectral* [5], [6], [7] or a *spatial* [8], [9] learning approach. The spatial GNN training is based on the *message-passing* paradigm and uses an iterative neighbourhood aggregation procedure to learn these low-dimensional embeddings [10].

GNNs use aggregated embeddings to calculate the updated embedding for a node or an edge. This process is performed on all nodes on a per-layer basis, in order to generate new embeddings for the whole graph per layer. There has been a rapid rise in spatial training approaches owing to their lower computational complexity. The forward pass in a *k*-layered GNN training recursively aggregates neighbourhood features that are *k* hops away from the *target* node, thus ensuring that the embeddings generated are representational of both the graph topology as well as the original node or edge features.

In this poster paper, we present a preliminary analysis of the computational cost of the forward pass on GraphConv and GraphSAGE GNN models. We verify our estimates empirically by profiling the forward pass on these architectures using graph datasets of varying sizes and properties. We observe a correlation between our analytical estimates and the observed CPU execution time, which validates the theoretical analysis. In future, this can be extended to the backward pass, and for distributed training of GNNs with communication costs.

† Student author

II. Background and Related Work

A theoretical analysis of the computational cost of GNN layers will assist in making compute-aware architecture choices and enable us to estimate the computation times of training for a given GNN architecture. Few works have performed this analysis.

Chiang et al. [11] present a study of the time and space complexities of a few standard GNN models assuming uniform intermediate embedding dimensions. They derive the time complexity of the standard GCN model to be $\mathcal{O}\left(L\|A\|_0F + LNF^2\right)$ and the GraphSAGE model to be $\mathcal{O}\left(r^LNF^2\right)$, where L is the number of layers, N is the number of nodes, $\|A\|_0$ is number of non-zeros in the adjacency matrix, F is number of features, and r is the number of sampled neighbours per node. Wu et al. [12] also perform a high-level time complexity analysis of the standard GCN layer, which comes out to $\mathcal{O}\left(E\right)$, for a graph with E edges. However, neither of these study the effect of different intermediate embedding dimensions and the number of output classes on the overall cost of training.

Such an analysis is also lacking for distributed training of GNNs, where communication and synchronization costs can play a role. E.g., distributed GNN training frameworks such as DistDGL [13] overlap the backward pass with synchronization. Our present study estimating the forward pass compute costs will help in such a setting, and assist in designing better distributed GNN training platform.

III. GNN Forward Pass Computational Cost

Here, we estimate the computational cost of different GNN layers in terms of the floating point operations (FLOPs) required to generate the embeddings for the next layer, for all vertices of a graph $G(V, E)$, where $|V|$ and $|E|$ represents the number of vertices and edges respectively. Each vertex i is assumed to have a feature/embedding vector $h_i^l \in \mathcal{R}^{d\times 1}$ of size d, which serves as an input to the l^{th} layer of the GNN. The output of layer l for the vertex i is denoted by the embedding $h_i^{l+1} \in \mathcal{R}^{d'\times 1}$.

Our FLOP calculations assume dense matrix operations within each layer. While these operations are optimized as sparse matrix operations for most frameworks, we see the trend hold even with this assumption. Next, we derive the calculation of the expected FLOPs for GraphConv and GraphSAGE.

GraphConv Layer. In its vanilla form, the GraphConv layer [1] uniformly averages the features (or the previous layer's embeddings) for the in-neighbours of the target vertex

979-8-3503-0208-0/23/$31.00 ©2023 IEEE
DOI 10.1109/CCGridW59191.2023.00076

Table I: Graph Datasets Used for Experiments

Graph	$\|V\|$	$\|E\|$	Feature Size (d_{in})	# classes ($\mathcal{C}$)
Cora	2.70K	10.56K	1433	7
Citeseer	3.32K	9.23K	3703	6
Amazon Photo	7.65K	238.16K	745	8
Wiki-CS	11.70K	431.72K	300	10
Amazon Computers	13.75K	491.72K	767	10
Coauthor CS	18.33K	163.79K	6.8K	15
Pubmed	19.71K	88.65K	500	3
Coauthor Physics	34.49K	495.92K	8.42K	5
Flickr	89.25K	899.76K	500	7
Yelp	716.84K	13.95M	300	100

to update its embeddings. Formally, the updated embedding h_i^{l+1} for vertex i generated by layer l is given as:

$$h_i^{l+1} = \text{ReLU}\left(U^l \frac{1}{\deg_i} \sum_{j \in \mathcal{N}_i} h_j^l\right)$$

where $U^l \in \mathcal{R}^{d' \times d}$ is the learnable matrix, $\text{ReLU}(x) = \max(0, x)$, $\mathcal{N}_i$ are the neighbours of node i, and deg_i is the degree of node i (for an undirected graph). This results in $((\deg_i + 1)\, d + 2dd')$ FLOPs per vertex per layer, where d and d' are the dimensions of the input and the output embedding vectors, respectively. For each GraphConv layer, aggregating the FLOP count over all vertices, we have:

$$\text{Expected FLOPs} = 2d\,|E| + (d + 2dd')\,|V| \qquad (1)$$

GraphSAGE Layer. The GraphSAGE (SAmple and aggreGatE) layer [14] improves upon the existing GCN and spectral models to incorporate the target vertex's old embedding to calculate the current layer's embeddings. Formally, the update equation of the GraphSAGE layer is:

$$\hat{h}_i^{l+1} = \text{ReLU}\left(U^l \text{concat}\left(h_i^l, \text{Mean}_{j \in \mathcal{N}_i} h_j^l\right)\right)$$
$$h_i^{l+1} = \frac{\hat{h}_i^{l+1}}{\|\hat{h}_i^{l+1}\|_2}$$

where $U^l \in \mathcal{R}^{d' \times 2d}$ is the learnable matrix and $\|\cdot\|_2$ represents the Euclidean norm. This results in $((\deg_i + 1)d + 3d' + 4dd' + 1)$ FLOPs per vertex per layer. So, the total FLOPs for each GraphSAGE layer across all vertices is:

$$\text{Expected FLOPs} = 2d|E| + (1 + d + 3d' + 4dd')|V| \qquad (2)$$

We extrapolate this to a 3-layer GNN model, assuming that the two inner layers are of sizes $d_{in} \times d_{in}$ and the output layer is $d_{in} \times \mathcal{C}$, where d_{in} is the input feature size and $\mathcal{C}$ is the number of output classes. The total FLOPs for the models are:

GNN Model	Expected Number of FLOPs
GraphConv	$6d_{in}\|E\| + \left(3d_{in} + 2d_{in}\mathcal{C} + 4d_{in}^2\right)\|V\|$
GraphSAGE	$6d_{in}\|E\| + \left(3 + 9d_{in} + 3\mathcal{C} + 4d_{in}\mathcal{C} + 8d_{in}^2\right)\|V\|$

IV. Empirical Evaluation

We use Eqns. (1) and (2) to estimate the FLOPs for the forward pass of various graphs (Table I) for these two layers, and compare them against their observed execution times.

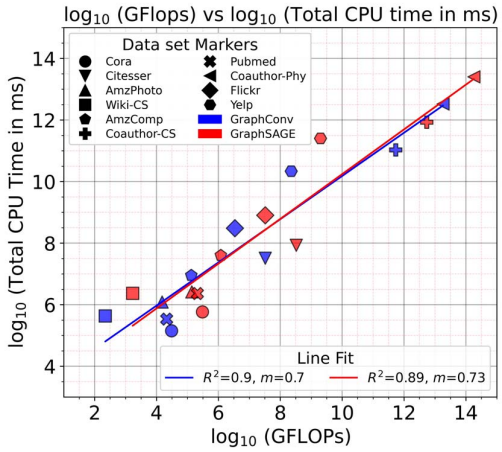

Figure 1: Log-log plot of *Expected GFLOPs* vs. *Observed CPU time (ms)*, with a linear fit of slope m, for the two GNNs.

Setup. We use the DGL framework (`dgl` v1.0.0) [15] with Pytorch as the backend to define the GraphConv and GraphSAGE models with 3 layers each. We run the experiments on an Intel Xeon CPU (E5-2620 v4 @ 2.10GHz) with 512 GB of memory. The training time is measured using the PyTorch profiler.

Datasets. We use 10 different graph datasets of varying sizes and classes, as shown in Table I, sourced from the `dgl.data` package. Besides being from diverse domains, they also vary in graph density between 5×10^{-5} to 8×10^{-3} and in feature vector dimensions (d_{in}) from 300 to 8415.

Results. Fig. 1 plots the total CPU time taken (Y axis, in ms) by a full graph forward pass computation for each graph dataset and the two GNN models, and correlate these against the estimate compute time from our analysis (X axis, in Giga FLOPs). The axes are in $\log_{10}$ scale. A linear trend line is fit for each model and its high R^2 implies a strong correlation between log of the expected FLOPs and log of the observed CPU execution time. While the line-fit should ideally have a slope of $m = 1.0$, the observed slope of $m \approx 0.7$ implies a strong correlation, particularly for larger values of expected FLOPs since, $f(x) = x^{0.7}$ can be approximated by a straight line for large values of x (which in our case is GFLOPs). This confirms the theoretical analysis presented in this paper.

V. Conclusion and Future Work

We have analytically derived the computational cost for the forward pass of two common GNN models, and find that they strongly correlate with the observed forwward-pass training times for diverse graphs. As future work, we intend to perform a similar analysis for the backward pass of GNN training, estimate the space (memory) complexity, and extend this to distributed GNN training. The analysis can provide a deeper understanding of the source of training costs, allowing us to develop better GNN training platforms.

REFERENCES

[1] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations (ICLR)*, 2017.

[2] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and leman go neural: Higher-order graph neural networks," in *AAAI Conference on Artificial Intelligence*, 2019.

[3] D. Wang, J. Lin, P. Cui, Q. Jia, Z. Wang, Y. Fang, Q. Yu, J. Zhou, S. Yang, and Y. Qi, "A semi-supervised graph attentive network for financial fraud detection," in *IEEE International Conference on Data Mining (ICDM)*, 2019.

[4] S. Wu, F. Sun, W. Zhang, X. Xie, and B. Cui, "Graph neural networks in recommender systems: a survey," *ACM Computing Surveys*, vol. 55, 2022.

[5] J. Bruna, W. Zaremba, A. Szlam, and Y. Lecun, "Spectral networks and locally connected networks on graphs," in *International Conference on Learning Representations (ICLR)*, 2014.

[6] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," arXiv, Tech. Rep. arXiv:1506.05163, 2015.

[7] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," *Neural Information Processing Systems (NeurIPS)*, 2016.

[8] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International Conference on Machine Learning (ICML)*, 2017.

[9] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," *Neural Information Processing Systems (NeurIPS)*, 2016.

[10] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *IEEE Data Engineering Bulletin*, vol. 40, 2017.

[11] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks," in *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019.

[12] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, 2021.

[13] D. Zheng, C. Ma, M. Wang, J. Zhou, Q. Su, X. Song, Q. Gan, Z. Zhang, and G. Karypis, "Distdgl: distributed graph neural network training for billion-scale graphs," in *IEEE/ACM Workshop on Irregular Applications: Architectures and Algorithms (IA3)*, 2020.

[14] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Neural Information Processing Systems (NeurIPS)*, 2017.

[15] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Li, and Z. Zhang, "Deep graph library: A graph-centric, highly-performant package for graph neural networks," arXiv, Tech. Rep. arXiv:1909.01315, 2019.