Efficient training of graph classification models

Hao-Yu Shih, Winfred Li, Noah Cohen Kalafut

October 25th, 2021

1 Introduction

Graph classification is a crucial problem with practical applications in many different domains [14]. The data from bioinformatics [7], chemoinformatics [10], social network analysis [4], urban computing [5], and cybersecurity [9] can all be naturally represented as labeled graphs. For instance, in urban computing, constructing effective bike lanes has become a crucial task for governments promoting the cycling lifestyle. The task is to predict and build well-planned bike paths that reduce traffic congestion and decrease safety risks for cyclists and motor vehicle drivers. Given the importance of graph classification, in this study, we are going to improve the efficiency of graph classification models' training.

1.1 Existing Solutions

There have been several attempts at achieving efficient computation in graph-based models, but not many that specifically focus on classification. One such example is [12]. This particular work details the motivation and creation of several distributed graph computation frameworks using a variety of graph partitioning methods. Although not specifically focused on applications that require advanced gradient synchronization methods, the author looks very thoroughly into the generality of the solution presented. For the purposes of graph classification, however, it is noted that various partitioning methods may trade in reproducibility in favor of efficiency. Additionally, the solutions presented are standalone in nature, and therefore may be limited in widespread usage.

With a greater focus on graph classification problems, [8] focuses on efficient training through transfer learning; specifically referring to embedding generation models. The pre-training steps are of particular interest here, as they provide a better start-point for the model and aid in its convergence. This work purely discusses the accuracy benefits, however, and doesn't attempt to parallelize the two training steps. A combination of distributed computing similar to [12] and compatibility with high-efficacy training techniques such as [8] could result in faster-training models for real-time applications or rapid prototyping. Integration with existing frameworks could also make this application more user-friendly which, as noted in [12], plays a large role in real-world usage of such frameworks.

1.2 Our Approach

We will focus primarily on implementing a maximally-efficient distributed computing solution for graph classification models. This solution will ideally have compatibility with the major graph classification models and training techniques discussed above. The solution should improve runtime on methods such as those in [8, 11] with little to no loss in accuracy. These results will be achieved through optimization on top of *DGL* [1]. Specifically, we will experiment with reduction of the number of graph partitions to find an optimum for tested graph classification models. This will allow for more throughput with less communication overhead. We will also experiment with changes to gradient synchronization frequency and aggregation techniques to find the most efficient solution for various graph classification problems. Another potential direction is to experiment with overlapping partitions where multiple partitions may contain common nodes. As before, the goal would be to reduce communication between machines. This could also be combined with the gradient synchronization techniques discussed earlier.

1.3 Expected Results

We intend to test our methods on semi-supervised node classification tasks. Potential datasets include citation networks such as Citeseer, Cora, and PubMed [15] as well as other datasets such as Reddit, PPI and Amazon [2, 3]. We plan to use Cloudlab in order to carry out the experiments.

2 Related Work

2.1 Neural Network Architecture

In [16], the authors designed the novel neural network structures that can accept graphs to train graph classifi-

cation models. They proposed a novel end-to-end deep learning architecture for graph classification and a novel spatial graph convolution layer to extract vertex features. The key innovation is a new SortPooling layer which takes as input a graph's unordered vertex feature from spatial graph convolution.The three aspects make their Graph Convolution Neural Network highly competitive compare with state-of-the-art graph kernels [6].

2.2 Training Methodology

There are many papers which focus on optimal convergence of graph classification models. Included among these are [11] and [8]. Between the two of these papers, several training techniques and architectures are explored for use in graph classification problems. In order to maximize compatibility, it would be ideal for our final efficient training solution to be able to overlap with these existing methods. Additionally, in [11], several best-practices are laid out for use in reporting graph classification model results. Adhering to these best-practices and encouraging replication studies will be an objective in this paper during reporting.

2.3 Existing Frameworks

Distributed computing and parallelization systems centered around graph classification models are somewhat rare. However, several papers discuss efficient graph computation. Among these papers is [12]. [12] creates several standalone applications for use in graph computation on distributed systems and is flexible enough to handle both synchronous and asynchronous workflows. However, these approaches are not specifically tailored to graph classification, and could be optimized using the above methods or using a similar methodology to *PipeDream* [13].

Additionally, we can look at existing frameworks for graph processing in order to get a sense of how these systems are designed. For instance, the DGL Python package [1] provides easy implementations of graph neural networks. It also provides support for distributed training. Their framework involves three processes: server, sampler, and trainer. The server process runs on each machine and stores a graph partition. The sampler interacts with servers to generate minibatches for training, and the trainers interact with the servers and samplers in order to carry out training.

3 Timeline and Evaluation Plan

3.1 Hardware and Software

During testing, we will be replicating author methodologies for the workflows described in each graph classification model's original paper. In addition, datasets such as Amazon [2], Citeseer, Cora, PubMed [15], PPI, Reddit [3] will also be utilized for efficiency comparisons.

For prototyping and development, we will use Cloudlab and host the code open-source on GitHub. For final runs on large graph classification models, Google Cloud may also be used. This is due to the long-running capabilities of Google Cloud.

3.2 Timeline

The below is a provisional outline of the project.

- Oct 29: Set up Framework Codebase
- Nov 1: Set up Cloudlab Instance
- Nov 12: Begin Framework Testing
- Nov 25: Begin Final Report
- Nov 30: Project Check-In Report Due
- Dec 7: Finish Final Report
- Dec 8: Start Presentation
- Dec 14: Finish Presentation
- Dec 20: Final Report Due

References

[1] `https://docs.dgl.ai/index.html`.

[2] `https://github.com/Hanjun-Dai/steady_state_embedding`.

[3] Graphsage: Inductive representation learning on large graphs. `http://snap.stanford.edu/graphsage/`.

[4] L. Backstrom and J. Leskovec. Supervised random walks: predicting and recommending links in social networks. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 635–644, 2011.

[5] J. Bao, T. He, S. Ruan, Y. Li, and Y. Zheng. Planning bike lanes based on sharing-bikes' trajectories. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1377–1386, 2017.

[6] K. Borgwardt, E. Ghisu, F. Llinares-López, L. O'Bray, and B. Rieck. Graph kernels: State-of-the-art and future challenges. *arXiv preprint arXiv:2011.03854*, 2020.

[7] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1):i47–i56, 2005.

[8] M. T. Do, N. Park, and K. Shin. Two-stage training of graph neural networks for graph classification, 2021.

[9] H. Duen, N. Carey, W. Jeffrey, W. Adam, and F. Christos. Polonium: tera-scale graph mining for malware detection. In *Proceedings of the SIAM International Conference on Data Mining (SDM)*, 2011.

[10] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. *arXiv preprint arXiv:1509.09292*, 2015.

[11] F. Errica, M. Podda, D. Bacciu, and A. Micheli. A fair comparison of graph neural networks for graph classification. In *International Conference on Learning Representations*, 2020.

[12] A. Guerrieri. Distributed computing for large-scale graphs. 2015.

[13] A. Harlap, D. Narayanan, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, and P. B. Gibbons. Pipedream: Fast and efficient pipeline parallel DNN training. *CoRR*, abs/1806.03377, 2018.

[14] J. B. Lee, R. Rossi, and X. Kong. Graph classification using structural attention. pages 1666–1674, 2018.

[15] P. Sen, G. Namata, M. Bilgic, L. Getoor, and B. Galligher. Collective classification in network data. *AI Magazine*, 29(3):93, 2008.

[16] M. Zhang, Z. Cui, M. Neumann, and Y. Chen. An end-to-end deep learning architecture for graph classification. 2018.