Overview of Full-Batch versus Mini-Batch Training of Graph Neural Networks

Andrew Tran
Yale University

Abstract

1 Introduction

Graph Neural Networks (GNNs) are a powerful class of machine leaarning models as they can operate on graphs, which are natural representations of a lot of data. Nowadays, with the advent of web-scale datasets, which can have millions of vertices and edges, this has prompted the need for more scalable GNN architectures and training methodologies. However, scaling has many challenges, especially due to the memory-intensive nature of GNN computations.

GNNs operate by performing rounds of message passing between neighboring vertices. This can be very memory-demanding and also makes caching/sharding data harder due to the less structured nature of data access in a graph. These access patterns also make it harder to fully utilize hardware accelerators, primarily GPUs, which are easier to optimize for more regular and dense computations. There have been many proposed methods for data management and model parallelization strategies to optimize for performance in light of this.

To address the challenges, there are two main categories of training techniques which are full-batch training and mini-batch training. With full-batch training, the entire graph is processed in each training epoch, allowing the model to capture the context of the entire graph at the cost of needing to address memory consumption and scalability concerns. On the other hand, mini-batch training divides the graph into more manageable subgraphs, allowing for easier scaling with tradeoffs such as not capturing all graph information. Each approach has distinct advantages and trade-offs with respect to performance and scalability.

We aim to provide a comprehensive comparison of full-batch versus mini-batch training methods. Recent empirical studies have evaluated these training methods. We also want to highlight the state-of-the-art techniques for both of these methods.

2 Full-Batch vs Mini-batch Training High-Level Overview

2.1 Challenges

There are inherent challenges to full-batch training of graph neural networks that are inherited from the general challenges of full-batch training of large models in general, especially on web-scale datasets. There is overhead in gradient synchronization. Full-batch training necessitates synchronizing gradients, which will be averaged, across many GPUs and nodes. This synchronization involves all-reduce operations, so the slowest GPUs and nodes can become substantial bottlenecks in the progress of the overall training. There is also increased communication overhead that scales up with the number of nodes which further reduces efficiency.

There is also the concern of memory intensity. For web-scale datasets, memory footprint easily can exceed the capacity of individual GPUs, or even the GPUs of a single node in aggregate. There is also more memory usage than standard models, especially for dense graphs with lots of connections where lots of message-passing occurs. This is part of the bigger problem of more irregular computation in GNNs which can make fully parallelizing the processing to fully utilize resources even more difficult. Finally, this is also related to the difficulty of complex data partitioning. It is non-trivial to partition the graph to balance the computational load while also minimizing the need for inter-partition communication for efficiency.

2.2 Performance

Studies have looked at the performance differences between full-batch and mini-batch training for GNNs. They highlight that the specific choice of training methodology can be very application-specific.

Bajaj [1] conducted an extensive peerformance comparison of full-graph and mini-batch training performance for GNNs. They found that mini-batch training was able to meet the performance of full-batch training for the models that they tested in most cases. However, this was with having the appropiate hyperparameters. This is in contention to what they say is a common assumption that full-graph training inherently preserves more information and will always have higher accuracy. However, they also highlight the sensitivity of each training method to hyperparameter settings. The hyperparameters that worked well in the mini-abtch setting did not necessarily also work well in the full-batch setting and vice versa, showing the nuance that hyperparameters have with these different methods to counter their tradeoffs. They do mention one big advantage of mini-batch training though, the ability to more effectively take advantage of efficient computing for higher performance.

Gasteiger [2] which introduced a method of mini-batching called Influence-Based Mini-Batching tries to address some of the shortcomings that they saw to mini-batch training. They stated that issues with mini-batch training included information loss and noisy gradients due to the stochastic nature of subgraph sampling, which loses global graph information. For example, imagine a case where random sampling of a local neighborhood randomly does not choose important edges for a prediction task. Gasteiger with their method attempted to optimize the selection of the mini-batches to avoid cases like these. They emphasize how naive mini-batching as used with Bajaj can sometimes already achieve great results relative to full-batch training. However, they show that realizing there is flexibility in sampling techniques opens the possibility of mini-batch training architectures.

Overall, the key takeaways of full-batch and mini-batch training for GNNs therefore can be summarized as

- **Performance/Efficiency** Mini-batch training if it can reach the desired accuracy can get there in less time due to the nature of it being easily parallelized. Therefore, it can be more efficient than full-batch training in general.

- **Accuracy** With proper hyperparameter tuning and better sampling techniques than naive random sampling, mini-batch training can often match the accuracy of full-batch training.

- **Scalability and Memory Usage** The scalability and reduced memory usage of mini-batch training comes from not needing to synchronize gradients and store a lot of information needed by other nodes.

- **Hyperparameter Sensitivity** While both methods are sensitive to hyperparameters, mini-batch may need more extensive hyperparameter search to offset the downsides of the nature of stochastically using only subgraphs of the entire dataset. Also, hyperparameters are not always directly transferable between the two training paradigms.

3 Methods

We will discuss a few state-of-the-art methods used for full-batch and mini-batch training.

3.1 Full-Batch Methods

[3] PipeGCN is a full-batch training method designed for efficiency. It partitions the graph into distinct subgraphs which are each allocated to separate GPUs. This is how memory constraitns are mitigated. Then, pipeline parallelism is utilized where different layers of the GNN are distributed across GPUs also. This allows for overlappign comptuation and communication to better utilize all of the hardware resources. It also makes sure to take advantage of high-speed interconnects like NVLink to reduce all of the overhead of communication necessary after already trying to minimize communication.

[4] In terms of inducing less structure on the end user, the Python Deep Graph Library (DGL) also provides a framework for full-batch training. It does so in a more generic nature than PipeGCN to allow for different methods of partitioning. Its main utility is allowing the library to do the hard part of the synchronization of model parameters and gradients with it providing a layer of abstraction above all-reduce.

3.2 Mini-Batch Methods

Influence-Based Mini-Batching (IBMB) was introduced by Gasteiger [2] and it addresses common issues including information loss and noisy gradients. Its approach emphasizes selecting mini batches based on the expected influence of the nodes on the output. Therefore, it tries to ensure that each mini batch includes nearby nodes to each target node for the task that will likely influence the outcome of the model. The idea is that this reduces the variance in gradients, helping convergence and stability. Empirical evaluation demonstrates that there are substantial improvements in training speed and inference efficiency from this.

GraphSAINT [5] is another mini-batch training method but it does not sample nodes and edges independently like traditional methods. It tries to sample subgraphs that maintain local connectivity, to better capture structural properties of the full original graph. Therefore, better neighborhood information is preserved for more accurate message passing in GNNs. The sho w that their method is effective for a diverse set of grap hdatasets.

4 Discussion

Therefore, there is a nuanced process to look at when deciding between full-batch and mini-batch training methods. Full-batch methods such as via PipeGCN and DGL's library support are able to fully capture global graph structures and can lead to better accuracy if global dependencies are critical.

However, they suffer greatly from scalability due to memory and communication throughput required. On the other hand, mini-batch addresses these two points directly but suffer from the random sampling of local neighborhoods in the graph which loses information. Methods such as IBMB and GraphSAINT try to reduce this gap to full-batch when it's present to achieve parity with full-batch training when the dataset is trickier for mini-batch. Therefore, the development of more sophisicated mini-batch techniques is very promising. It will be interesting to see how the general usage of full-batch training and more sophisicated mini-batch training change with more complicated offerings in both paradigms.

References

[1] Saurabh Bajaj et al. *Graph Neural Network Training Systems: A Performance Comparison of Full-Graph and Mini-Batch*. 2024. eprint: arXiv:2406.00552.

[2] Johannes Gasteiger, Chendi Qian, and Stephan Günnemann. *Influence-Based Mini-Batching for Graph Neural Networks*. 2022. eprint: arXiv:2212.09083.

[3] Cheng Wan et al. *PipeGCN: Efficient Full-Graph Training of Graph Convolutional Networks with Pipelined Feature Communication*. 2022. eprint: arXiv:2203.10428.

[4] Minjie Wang et al. *Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks*. 2019. eprint: arXiv:1909.01315.

[5] Hanqing Zeng et al. *GraphSAINT: Graph Sampling Based Inductive Learning Method*. 2019. eprint: arXiv:1907.04931.