Scaling Graph Neural Networks for Drug-Drug Interaction Prediction Using Partitioning and Parallelization

Farzaan Kaiyom
Department of Computer Science
Stanford University
farzaan@cs.stanford.edu

Abstract

The increasing size of machine learning datasets has raised new questions of scalability. This problem is even more difficult in the more specific field of machine learning with graphs, because graphs have interconnected data that can't be arbitrarily split to parallel workers. Accordingly, sampling based approaches have provided a way to scale. However, sometimes, especially within the clinical domain, using an entire dataset is far preferable. This is because graph sampling has been shown to damage model performance by a non-negligible amount and it can be expected that ignoring any drugs or proteins in a clinical machine learning model could become problematic. This paper proposes a method to train graph neural networks without relying on sampling. Graph partitioning provides a way to parallelize workers in a way that reduces the cut between subgraphs. This paper investigates this approach's efficacy in the more specific problem of drug side effect prediction, a task where datasets have grown increasingly large and sampling has been a common previous approach to address data size. The methods shown in this paper show scalability similar to that of sampling, without sacrificing as much accuracy. Accordingly they provide a way to train clinical machine learning models for side effect prediction with greater ease and efficiency.

1 Introduction

The problem investigated in this work is scalability for the task of drug-drug interaction prediction. Drug-drug interaction prediction involves using drugs, and some of their relations to other drugs (and often times other biomedical entities) to predict unseen relations (often adverse side effects) between drugs as a binary classification task. This particular problem is interesting because it has been shown that models for drug-drug interaction prediction have succeeded in predicting new relations that have been validated by medical research. The prediction of new relations using neural networks can thus accelerate the process of drug side effect discovery Zitnik et al. (2018). This is of great importance because some drug-drug interactions are fatal or have irreversible bodily effects. The use of machine learning for drug interaction prediction has actually allowed for researchers to conduct drug research with greater awareness of expectations which has led to increased efficacy and potentially increased safety.

This paper looks more specifically at scalability of predicting drug-drug interactions because machine learning with graphs itself is very computationally taxing especially when increasing amounts of data. A common approach to address scalability in machine learning with graphs has been sampling, but sometimes graphs are too large to fit on a GPU so even sampling approaches don't suffice. Even if data can fit on a machine, sampling doesn't provide a perfect solution since it creates non-negligible

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

error through approximation Jia et al. (2020). While this is acceptable in other domains like graph neural networks for product or video recommendations, in the high impact clinical domain it is not. It's straightforward that ignoring some drugs or proteins in a clinical machine learning model could have potentially damaging effects on a downstream prediction task. Additionally training of machine learning models in this domain can take a very long time (hours-days), especially when using small batches. This, along with a historical lack of data, could explain why previous work has only worked with datasets of a few thousand drugs and proteins. This work strives to test the scalability and performance of a neural network on much larger and comprehensive datasets. Scalability is a particularly difficult challenge with graph formatted data, because it requires smart partitioning of data to workers, since the data cannot be arbitrarily split as it is interrelated. Finding an intelligent way to partition data and parallelize training of neural networks for drug-drug interaction is important because it can help speed up research on drugs side effects and allow for the full use of datasets that previously could not be used without sampling.

This paper proposes a parallelized approach that doesn't require sampling to train a graph neural network. By partitioning a graph to multiple workers and then training them in parallel, the proposed method can scale without needing to approximate the data distribution. Results show that this successfully scales nearly as well as sampling, with some small drawbacks, but with far greater accuracy.

2 Related Work

The work in this paper is related to previous work in two intersecting sub-topics: message passing graph neural networks and drug-drug interaction prediction. It's important to understand both the foundations of the models discussed in this paper and the nuances of the application they'll be used for, so here is a discussion of both of these topics and their relationships in the aforementioned order.

2.1 Graph Neural Networks and Scalability

Graph neural networks are a specific type of neural network that works specifically with graph structured data (i.e. nodes and edges). This often involves finding the best way to encode the information of a node using its nearby neighbors. A good example is the approach taken by Kipf and Welling (2016), graph convolutional networks. These can be compared to nromal convolutional neural networks and their application to image related tasks: by looking at the neighborhood subgraph or convolution around each node and aggregating the feature vector into that node, graph convolutional networks provide an intuitive way to represent nodes in graphs. This gives every node an embedding that can be used for downstream tasks like node classification and link prediction. The GCN model popularized the notion of graph neural network and is in many ways the simplest GNN architecture. Accordingly it will serve as a baseline in this work for accuracy and scalability.

The GCN approach is built upon in Hamilton et al. (2017), with explicit emphasis on scalability. Their approach is somewhat identical except that it involves using sampling to increase runtime and efficiency. More specifically, when generating neighborhood subgraphs for each node, it only considers a sampled subset of nodes at each step. While this may have potential accuracy drawbacks due to its ignorance of non-sampled neighbors, it makes it feasible to train graph neural networks on graphs with millions or even billions of nodes and edges. It achieves unprecedented scalability with little shown consequences on accuracy. GraphSAGE serves as an approximated scalable method for graph neural networks that can be compared against the methods proposed in this paper.

Another approach that looks at scaling graph neural networks is described in Chiang et al. (2019). They propose ClusterGCN, a method that uses common partitioning algorithms like METIS to identify connected clusters within a graph to generate and train mini-batches within. This provides yet another way to scale graph neural networks which is exactly what this work seeks to do. This does, however, have the drawback of sometimes cutting off neighborhoods for nodes on the edge of each partition, but again this doesn't stop it from achieving its goal of providing a more scalable approach to graph neural networks. This is one primary difference between ClusterGCN and the proposed method of this paper; the proposed partitioning method creates overlapping partitions so each node unique to a partition has its full neighborhood. Another difference is that ClusterGCN successively trains these partitions on a single worker, whereas the proposed method by this paper scales to multiple methods.

While the aforementioned methods are strong ways to scale GNN training when a graph can fit in memory, graphs don't always fit in memory. Further, each of the mentioned methods require a sort of sampling or cutting of some nodes or edges. Kibata et al. provide a solution to these concerns. Their approach involves creating partitions where each node has its entire neighborhood, which means some partitions overlap. Then multiple GPUs simultaneously train solely on each partition and share their gradients to train a consistent model. Their approach takes inspiration from Chiang et al. (2019)'s partitioning but is different in that avoids the problem of cutting off neighborhoods and also parallelizes training of each partition. This sped up runtime of a GCN by 3.28 by using 4 GPUs, an almost perfectly linear increase in performance per GPU. Zeng et al. (2020) takes a very similar approach of partitioning a graph into partitions for each worker and then training in parallel. The main distinction between these two works is the approach to partitioning; Zeng et al. (2020) uses a partitioning algorithm to optimize memory performance while Kibata et al. uses a partitioning algorithm that optimizes for balancing edges within each partition. These partitioning and parallelization approaches have been shown to consistently outperform accuracy of sampling methods like GraphSAGE while providing similar scalability (Jia et al. (2020)).

These partitioning and parallelization approaches are most similar to the approach proposed by this paper by far. The only difference is the partitioning algorithm used, and the application. Rather than using performance based partitioning, this paper proposes graph balancing partitioning like METIS. And of course, the most important difference between this work and the previous work is the difference in application. While accuracy is slightly better for the partitioned approach than in the sampled approach for generic web user prediction tasks like those in the aforementioned papers (Reddit community prediction), the disparity between sampling and partitioning has yet to be seen in the clinical domain, where one would expect sampling would do even more poorly compared to the partitioning approach. This is because ignoring some users in a user graph can work because there might be similar users nearby those users, which is the assumption that GraphSAGE's sampling hinges on Hamilton et al. (2017); however, this assumption doesn't necessarily hold up in a biomedical knowledge graph where each protein and drug has independent signficance. That's why it's important to see how all of these methods hold up (in scalability and accuracy) for the specific task of drug-drug interaction prediction.

2.2 Drug-Drug Interaction Prediction

The aforementioned methods are generic to any graph structured data, so its important to separately discuss the specific domain of drug-drug interaction prediction.

Zitnik et al. (2018) applies Kipf and Welling (2016)'s graph convolutional networks to the task of predicting polypharmacy side effects (i.e. side effects between multiple drugs). This paper uses information about protein-protein interactions as well as drug-protein interactions to predict drug-drug interactions. In order to take this into account, the GCN in this paper, called Decagon, encodes every edge type differently (protein-protein vs protein-drugs for example). This works well for the task. This paper's task is identical to the one I'm investigating. One difference is that this paper evaluated a model on a much smaller dataset. One with only about 19 thousand protein nodes. Accordingly the model is only trained on a single worker, so no parallelization is involved.

Nováček and Mohamed (2020) explicitly builds upon Zitnik et al. (2018)'s Decagon with a different approach. Rather than using graph neural networks, this paper uses knowledge graph embedding generation, a matrix optimization method that embeds components of a knowledge graph into vector space. These vectors then are used to rank potential subject-object-predicate triplets that represent potential links. This paper achieved slightly better AUROC scores than Decagon, but again didn't investigate the scalability of their approach.

Yu et al. (2020) provide a method that is a sort of mix between the methods in Nováček and Mohamed (2020) and Zitnik et al. (2018). They provide a graph neural network approach that also takes into account knowledge graph embeddings, effectively getting the best of both worlds. Additionally this model was evaluated on DrugBank data, like my model will be. However, this experiment used a DrugBank graph from 2008 which is, edge-wise, 10 percent the size of the one I'm using (from 2018). Again, as a result, this paper has no look into scalability.

Malone et al. (2018) can be thought of as a predecessor to Nováček and Mohamed (2020). It also investigates using knowledge graph embedding optimization for drug-drug interaction prediction.

Similar to its successor, this work doesn't investigate parallelization or scalability. Further, it didn't outperform Decagon. In fact in Yu et al. (2020), this model is shown to perform worse than any of the aforementioned ones. This could perhaps suggest that message passing graph neural networks may be important for drug-drug interaction prediction.

Lu et al. (2017) is another older paper that looks at drug-drug interaction without using graph neural networks. It takes the simple approach of using simalarity indices between nodes and their targets to predict whether they have an edge between them. It evaluates a number of different similarity measures including Jaccard similarity and Katz index. The results are decent but not as good as Zitnik et al. (2018) for example, perhaps again suggesting that message passing graph neural networks or even knowledge graph embedding generation are better methods for this specific task. These older, non graph methods have little relevance to the approach proposed in this paper, aside from the fact they involve the same ultimate prediction task.

3 Data

The dataset used in this project will be the BioSNAP knowledge graph, the largest publicly available knowledge graph of biomedical data. It was compiled by the SNAP Group at Stanford University and contains a plethora of biomedical relational data from various reputable sources. It is a heterogenous graph, containing a number of different node types representing different biomedical entities. For the selected problem, bipartite subgraphs have been extracted from this graph involving drugs and their relations to proteins, genes and other drugs. These subgraphs involve data collected from DrugBank (for drugs), STRING (for proteins), GeneOntology and HUGO (for genes). While the reason for including the drug graphs is straightforward, the reason for including the protein and gene graphs is a bit more complex. As noted before, previous work has established the relevancy of protein information for drug interaction prediction. The reason for this is that drugs usually have gene and/or protein targets, and these targets can provide information about drug-drug interactions. In this case we are examining the gene targets for each drug and the proteins that those genes create. The message passing neural network will help us better represent the drugs because it will take into account their gene and protein targets and the relations between those targets and other genes/proteins in addition to the drugs themselves. The bipartite supgraphs are sized as follows:

Bipartite Subgraph	Num. Edges
Drug-Drug	2,712,183
Drug-Gene	20,644
Gene-Protein	18,650
Protein-Protein	1,144,563

Here is an example of the data: in the graph there is an edge between the drug, Acetaminophen, and the drug, Acemetacin, because 'The risk or severity of adverse effects can be increased when Acetaminophen is combined with Acemetacin.' This provides an example of an edge in the graph representing a relation between two drugs and also conveys why these relations are important: they can have real adverse effects on patients. It would be difficult to visualize this graph or even a subgraph within it, as the average degree is 200, with many nodes having a degree of 700-1000. Since the graph is so densely packed with edges and large, it's difficult to visualize, but this statistic can provide an idea of how it looks. Common drugs that have widespread interactions with other drugs like Acetaminophen a common painkiller, for example, have thousands of neighbors in the Drug-Drug subgraph alone. Meanwhile there is a decent amount nodes in the graph that have degree of 0.

4 Approach

The approach proposed by this work have 3 main components: first the partitioning of a graph into pieces for each worker, then the parallelization of workers using distributed software, and finally the parallelized training of a graph neural network.

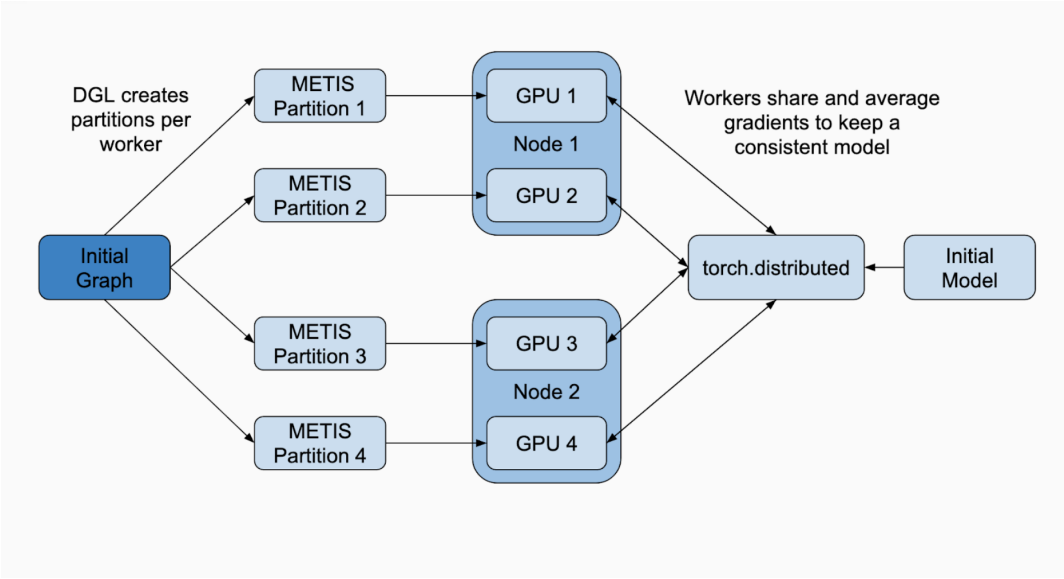

Figure 1: Parallelization Approach

4.1 Partitioning

In order to utilize multiple workers, one must partition a graph into pieces as is done in Kibata et al.'s work. Since Chiang et al. (2019) has established the efficacy of traditional graph partitioning methods like METIS for GNNs, this paper uses METIS to create partitions for each worker rather than a performance optimizing approach. This means the partitioning algorithm will prioritize balancing nodes and edges between partitions over runtime and efficiency. In contrast to Chiang et al. (2019)'s ClusterGCN, the method used in this paper includes overlapping nodes, so that each node that is unique to each partition has its entire 2-hop neighborhood within that partition. This means that each partition has a large amount of nodes unique to that partition, but also has non-unique/overlapping nodes that represent the near neighbors of those unique nodes that might be unique to other partitions. This ensures that when computing the embedding/representation of each node, no neighbors are cut off.

4.2 Parallelization

After partitioning, the approach is to use torch.distributed with an NCCL backend for synchronization of workers. This allows for one easy AllReduce call that ensures all workers have the same averaged gradient. This approach is similar to that of Kibata et al.. Each worker only has to deal with its own partition, and will synchronize its gradients with the other workers every epoch or every k epochs. In this work we set k to be 3, for the sake of simplicity and so that there isn't too much communication and so that the workers don't have much time to diverge. The synchronization process ensures that after every few epochs, each worker has the same model although they are training on different parts of the graph. After partitioning and parallelizing, we observed accuracy and F1-scores identical to that of a baseline without partitioning and significantly reduced memory usage and runtime compared to the baseline. The full parallel synchronization flow is visualized above.

4.3 Graph Neural Network

Regarding neural network archictecture, my approach is to take a graph convolutional network model, the most basic of all graph neural networks, train it on drug-drug interaction data, and evaluate its accuracy, runtime, and memory usage when combined with different scalable methods. More specifically, we compare the baseline single worker GCN approach to partitioned and parallelized approaches utilizing 2 and 4 workers. We also compare these approaches to the scalability and accuracy of GraphSAGE's neighbor sub-sampling, which will serve as a separate scalable method to compare the partitioning approach against.

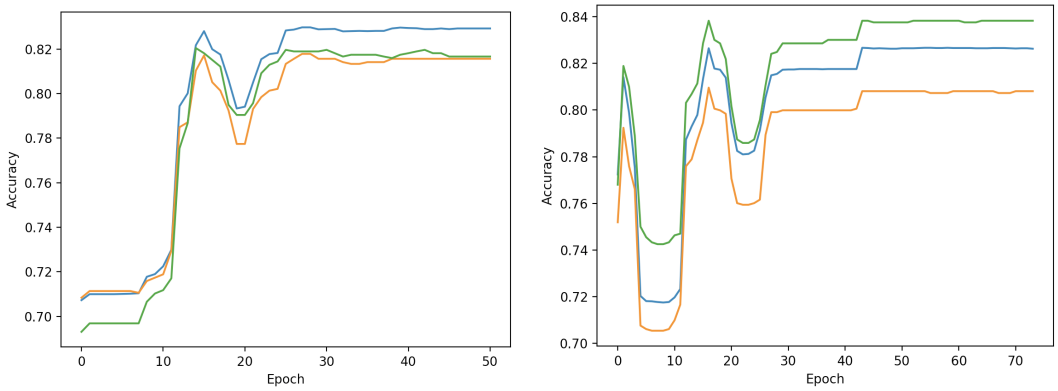

Figure 2: Training Plots from Baseline (No Sampling or Partitioning) versus Partitioned (x2)

5 Experiments

5.1 Experimental Setup

As mentioned before there were 4 experiments ran: a baseline single worker GNN (graph neural network) with no sampling or partitioning, a 2-worker GNN with parallelized partitions, a 4-worker GNN with parallelized partitions, and a GNN using GraphSAGE neighbor sampling. The latter 3 methods will serve as scalable methods to compare in accuracy and scalability to the baseline GCN approach. The GraphSAGE sampling has been made to approximately scale similar the 2-partition approach, so there can be a partitioned experiment to easily compare its accuracy against. To ensure that the GraphSAGE experiment scaled similar to in the 2-partition experiment, the model was made to sub-sample up to 100 neighbors for each node. This is because it had to reduce memory usage and runtime by about 50% to come close to the those of 2-partition approach. I selected 100 neighbors because each node in the graph had an average degree of 200, so by sampling up to 100 on average would have comparable scalability to the 2 partition approach. As a result the GraphSAGE's accuracy could be directly compared agaisnt the 2-partition approach in order to understand the differences in their scalability-accuracy tradeoffs.

The previous section explains the implementation at large, but not the specific software and hardware used. The partitioning algorithm used was METIS; a more specific METIS implementation that allows for overlapping neighbborhoods for GNN training has been used from DeepGraphLibrary (linked in the GitHub repository for this project). The neural networks in this project were implemented in Pytorch Geometric for 2 primary reasons. First, Pytorch Geometric is built on top of PyTorch which provides easy interfacing with torch.distributed for distributed communication and synchronization as needed in the project. Second, Pytorch Geometric provided precompiled modules for GCN and GraphSAGE in the GCNConv and SageConv modules. As mentioned before, within torch.distributed NCCL (NVIDIA Communications Library) was used as a backend to communicate between the NVIDIA GPUs used. This brings up the hardware used: 4 Nvidia Quadro RTX 8000 48GB GPUs, 2 Xeon E5-2623 v4 CPUs, and 2TB of RAM from a computing cluster generously provided by the SNAP group at Stanford University.

5.2 Results

On this page and the following page are training plots showing model performance (train, test, and validation) across training epochs. Each figure shows the baseline's training plot compared to one of the scalable approaches (either GraphSAGE or the proposed partitioning approach). This allows for easily comparison of training accuracy across epochs and model efficacy across time. The shown figures are for the 2 partition approach and the GraphSAGE approach, because as explained before these two approaches have about the same scalability, but different accuracy.

In the first plot (on this page), we can quickly note that the baseline and partitioned approaches have similar accuracy and somewhat similar training plots. This is because the partitioned approach is not an approximation like GraphSAGE; it is taking into account every node, just on separate workers.

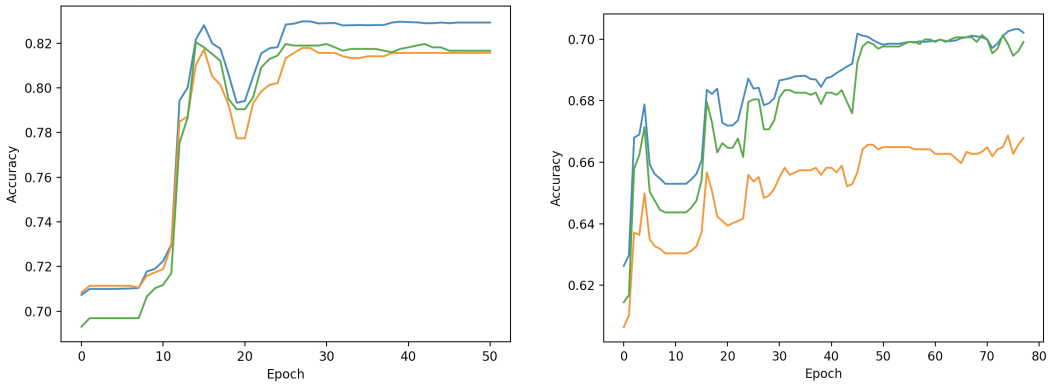

Figure 3: Training Plots from Baseline (No Sampling or Partitioning) versus GraphSAGE Sampling

The difference between the partitioned approach and the baseline is that the partitioned approach needs to synchronize the model between its workers every 3 epochs, which could explain it's more erratic nature. Regardless, it converges to a very close train accuracy to that of the baseline, which is impressive since it provides significant scalability through higher supported dataset size and lower runtime, as will be shown later in this work.

In the second training plot comparison, one can observe that GraphSAGE ends up converging at a lower accuracy than the baseline and is even more erratic than the partitioned approach relative to the baseline. This is because it is only considering some neighbors of each node. This could explain its low converged accuracy. As mentioned before there are lots of super-nodes in this graphs, or nodes with high degree. There are a few hundred nodes with 700-1200 neighbors. One can see why only considering 100 of these super-nodes' neighbors could negatively affect their representation's and accordingly the model's accuracy. The sampling provides both a potential explanation for both the choppiness of the training plot and a potential explanation for its poor accuracy relative to the baseline.

The following are some more specific numeric metrics representing the scalability of each approach and the according performance of each model. They can provide insights into just how much scalability is provided by the scalable methods investigated in this paper. They also allow these scalability metrics to be directly compared against performance relative to the baseline. Accuracy and F1-score serve as straightforward performance metrics for the model. Runtime, measured as time per epoch, is a scalability metric because reduced runtime means models, regardless of size, can be trained and tuned more quickly. Memory usage is a scalability metric because it serves as a way to quantify the supported dataset size. For example if memory usage went over 100% with some large graph in the baseline approach (which would crash the training program), it may not go over 100% and thus be supported with the scalable methods if they can reduce memory usage enough relative to the baseline.

	Baseline	2 Partitions	4 Partitions	GraphSAGE Sampling*
Avg GPU Memory Usage	35.63%	19.12%	11.93%	17.81%
Avg Time per Epoch	0.602s	0.337s	0.210s	0.283s
Accuracy	0.814	0.809	0.814	0.669
F1-Score	0.801	0.803	0.800	0.712

One can note again that the partitioned approaches have comparable accuracy to the baseline, while GraphSAGE has significantly lower accuracy. However, it's not all bad news for GraphSAGE, as GraphSAGE seems to scale more cleanly than the partitioning approaches. GraphSAGE cleanly splits memory usage and runtime in half, while the partitioning approaches don't (or in the 4 worker case partitioning fails to split memory usage into fourths). This is because the partitions aren't cleanly cut, and are overlapping to include neighbors of unique nodes to each partition. This overlap means that memory usage doesn't cut perfectly in half. Further, communication between workers after every few epochs serves as a further bottleneck in the partitioned approach. So GraphSAGE sampling provides slightly better scalability compared to the proposed partitioning and parallelization

approach, although it sacrifices a significant amount of accuracy to do so. When considering the full scalablity-accuracy tradeoff, if one wants to scale without sacrificing accuracy for drug interaction prediction, parallelization and partitioning appears to be the best method.

6 Conclusion

The experiments show that partitioning and parallelization provide scalability nearly as good as sampling approaches, without sacrificing nearly as much accuracy. However this was already shown by Jia et al. (2020) and Kibata et al. to some extent. The more specific insight that is provided by this work is as follows. These results provide the key insight that partitioning is particularly more accurate than sampling in the biomedical sphere. While Jia et al. (2020) noted some discrepancies in accuracy between GraphSAGE sampling and parallel partitioning when used for social network modeling, we have observed a major discrepancy when using these methods for drug-drug interaction prediction. This can be explained by the fact that assumptions that neighbor sampling may hold more in social graphs, where many users are similar and provide similar insights, than they do in biomedical graphs, where each node (a biomedical entity like a drug or protein) has a unique individual significance. This means that indeed, sampling is not ideal when using graph neural networks in the biomedical sphere to model biomedical data.

While this may seem to suggest that partitioning and parallelization should replace sampling completely, this is not the case. As noted before, the key insight here is that sampling is particularly bad for biomedical data. Sometimes its approximations work nearly as well as partitioning or baselines without sampling Jia et al. (2020). Sampling can provide a method to scale when a slight decrease in accuracy is acceptable and neighbor approximation would work well. Good examples of places where sampling with graph neural networks would work well includes any work with social network graphs where similar users are usually nearby (Hamilton et al. (2017)). Sampling could provide acceptable results as a method to scale in these domains. However the results of this paper show that parallelized partitioning is far better for biomedical tasks, or at least the task of drug-drug interaction prediction, where ignoring a few drugs and proteins could damage a model's predictive power.

One important note regarding this work is that it does not strive to outperform previous work at the task of drug-drug interaction prediction (although the baseline and partitioned models surprising perform within 0.1 of the state of the art). Instead the goal of this work is to compare the effects of different scalable methods on the accuracy of a simple baseline model, specifically for the drug-drug interaction prediction task. This is similar to the goal of Jia et al. (2020) where investigating the scalability-accuracy tradeoff between multiple methods was investigated; rather than trying to outperform previous methods, this work seeks to analyze how different approaches to scaling these methods may affect accuracy. This work seeks to do exactly this, except in the more specific domain of drug-drug interaction prediction. Indeed this work does, by showing that partitioning and parallelization serves as the strongest method to scale graph neural networks while sacrificing the least accuracy.

Future work could further investigate how partitioning can improve the accuracy of scalable approaches to other similar tasks. Drug discovery, for example, is a growing field that often uses graph neural networks and could benefit from scalable approaches that don't sacrifice accuracy. Additionally future work could strive to address the scalability shortcomings of the parallel partitioning approach. The communication bottleneck means that parallel partitioning doesn't scale as perfectly well as sampling. If future work could reduce or alleviate the communication costs of parallelized training, it could effectively make parallelized partitioning the best scalable approach to training graph neural network by all metrics. Until then, however, parallelized partitioning can provide a scalable approach that can decently outperform sampling in accuracy, but can't scale as perfectly as it.

7 Additional Information

7.1 Contributions

Farzaan Kaiyom contributed the code and writing of this project, with specific mentorship from Joy Hsu (TA for BIODS 220 @ Stanford University) and guidance from the rest of the instructors and TAs of BIODS220 and BIODS388 at Stanford University in Fall of 2020.

7.2 Codebase

The code used for this project can be founds at https://github.com/farzaank/scalableGNNbioSNAP

7.3 Regarding BIODS 220 and BIODS 388

With instructor consent I've enrolled in both BIODS 220 and BIODS 388. This means I am using the same project for both classes, with this paper representing a technical paper while the other class's final represents a non-technical clinical study. Accordingly I've attached the paper for the other class in the supplementary materials. The primary similarities are small and can be found in the figures and data analysis, while the differences are widespread and found in the analysis and explanations.

References

Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. 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.

Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034.

Zhihao Jia, Sina Lin, Mingyu Gao, Matei Zaharia, and Alex Aiken. 2020. Improving the accuracy, scalability, and performance of graph neural networks with roc. *Proceedings of Machine Learning and Systems (MLSys)*, pages 187–198.

Tokio Kibata, Mineto Tsukada, and Hiroki Matsutani. An edge attribute-wise partitioning and distributed processing of r-gcn using gpus.

Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Yiding Lu, Yufan Guo, and Anna Korhonen. 2017. Link prediction in drug-target interactions network using similarity indices. *BMC bioinformatics*, 18(1):1–9.

Brandon Malone, Alberto García-Durán, and Mathias Niepert. 2018. Knowledge graph completion to predict polypharmacy side effects. In *International Conference on Data Integration in the Life Sciences*, pages 144–149. Springer.

Vít Nováček and Sameh K Mohamed. 2020. Predicting polypharmacy side-effects using knowledge graph embeddings. *AMIA Summits on Translational Science Proceedings*, 2020:449.

Yue Yu, Kexin Huang, Chao Zhang, Lucas M Glass, Jimeng Sun, and Cao Xiao. 2020. Sumgnn: Multi-typed drug interaction prediction via efficient knowledge graph summarization. *arXiv preprint arXiv:2010.01450*.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2020. Accurate, efficient and scalable training of graph neural networks. *Journal of Parallel and Distributed Computing*, 147:166–183.

Marinka Zitnik, Monica Agrawal, and Jure Leskovec. 2018. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics*, 34(13):i457–i466.