Reproducibility report for Identity-Aware Graph Neural Networks

Zhihao Dong
School of Computing
Queen's University
Kingston, ON
z.dong@queensu.ca

Jeremy Banks
School of Computing
Queen's University
Kingston, ON
jeremy.banks@queensu.ca

Reproducibility Summary

Scope of Reproducibility

The authors of the paper claim that the proposed model, Identity-aware Graph Neural Networks (ID-GNNs), has greater expressive power than the existing GNNs, allowing it to distinguish between cycles of different lengths. Existing GNNs can achieve on average 25% accuracy improvement on node classification tasks by transforming it with the ID-GNN extension. The improvement becomes smaller on real-world dataset benchmarks, only 1.3% on node classification tasks[1].

Methodology

We used the GraphGym platform provided by the authors to allow for quick iteration and consistency through all of our experiments. This allowed us to quickly iterate through experiments comparing the results from PyTorch and TensorFlow implementations. Certain aspects of the platform made it struggle to run on personal workstations. Thankfully, we had access to a robust development environment. Building an anaconda virtual environment on the development environment allowed the dependencies to function correctly and produce results similar to those seen in the original author's paper. We implemented a total of 12 different models and tested them each across 4 different datasets in the node classification task.

Results

We found similar results to those in the original paper. Our results generally provided a much better improvement over the models of the original paper, with an overall improvement of 32.7% compared to their 24.9%.

What was easy

The requirements for running their code are fully outlined in the README.md file on GraphGym's GitHub page. Our initial attempt to install and execute GraphGym was outside of a virtual environment. There were some problems with dependencies that we did not dig further into. Running a Conda virtual environment and following the exact steps outlined in the readme allowed us to get an environment capable of executing the author's original code.

What was difficult

The complexity of the GraphGym platform made code comprehension very difficult. We had originally planned to fully integrate our code with their system to provide easy access between each deep learning framework (PyTorch and TensorFlow) when running experiments. While the platform was designed to be extensible, it was only extensible from the perspective of adding new architectures, not from the perspective of using different frameworks.

Communication with original authors

To date, we have not had any communication with the original authors.

33rd Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

1 Introduction

Graph Neural Networks (GNNs) have achieved great success in representing graphs for machine learning tasks. There are two main directions that GNNs take, which are spectral-based approaches and spatial-based approaches[2]. Spatial-based approaches[3, 4, 5, 6] are more commonly used because they are simple, efficient and have strong performance in real-world applications. Messaging passing is a central idea behind spatial-based GNNs, which allows the model to learn node representations through the aggregation of information from local node neighborhoods[7].

However, existing GNNs are limited in their ability to distinguish similarity between two graphs, or rather, if they are isomorphic. A common test for graph isomorphism is the 1-Weisfeiler-Lehman (1-WL) test, which represents the current upper-bound for GNN usage. Other work has been completed to improve GNN performance in domain specific scenarios and they all require a higher computational cost. Identity-aware Graph Neural Networks (ID-GNNs), proposed by the paper we are reproducing, are an attempt to create a generalized solution to apply in any graph scenario[1]. ID-GNNs apply heterogeneous message passing to replace the existing plain message passing to overcome the upper bound of the 1-Weisfeiler-Lehman (1-WL) test. The heterogeneous message passing is achieved by the Identity-aware (ID) mechanism. The core of the proposed Identity-aware (ID) mechanism is to distinguish the central node and its local neighbouring nodes using different transformation metrics. This paper claims that their proposed augmentations perform beyond the restrictions from the 1-WL test, and function across all styles of graphs.

2 Scope of reproducibility

The original paper sets about finding a way to increase the ability of a GNN to discern how to represent a graph.

The paper makes the following claims:

- ID-GNNs outperform classical GNNs
- Identity information can be formed from cycle count without adding to the computational complexity of the neural network.
- ID-GNNs perform better on synthetic datasets than they do on real datasets
- ID-GNNs provide a minimum of 1% improvement over non-augmented GNNs
- ID-GNNs provide an average of 25% improvement over non-augmented GNNs

3 Methodology

Our initial plan was to re-implement their code near-verbatim in TensorFlow. This approach was extraordinarily daunting and caused us to get lost in tracing through dictionary calls to figure out which pieces of code were to be called next. Instead, we decided to write a majority of the code from scratch, not concerning ourselves with the minutia of optimizer's weight decays and scheduler steps.

We maintained portions of their code related to the YAML configuration files in order to allow us to quickly iterate through different experiments and relate our values to theirs. We also had access to GPUs for running tests faster, but this wasn't required. We made heavy use of the tf_geometric python library[8], which is a library for graph-related neural network processing similar to PyTorch Geometric. PyTorch Geometric was the framwork used by the original authors. We used a single Nvidia RTX 2080 Ti with 11GB of RAM to train all of our models.

3.1 Model descriptions

The paper we are reproducing extends existing GNN architectures by inductively considering nodes' identities during message passing, named 'heterogeneous message passing'. In this section, we simply introduce the existing GNN models that can be extended by the heterogeneous message passing. Then we illustrate the Identity-aware mechanism proposed by this paper to achieve the heterogeneous message passing.

3.1.1 Classical GNN Architectures

The following is a list of the most common GNN architectures used in machine learning for graphs. These models are based on a form of message passing in which messages are exchanged between nodes, shown in Figure 1. During the message passing in a GNN, node representations will be updated by aggregating information from its local neighbourhood. The message passing update of the l-th layer in a GNN can be expressed as follows:

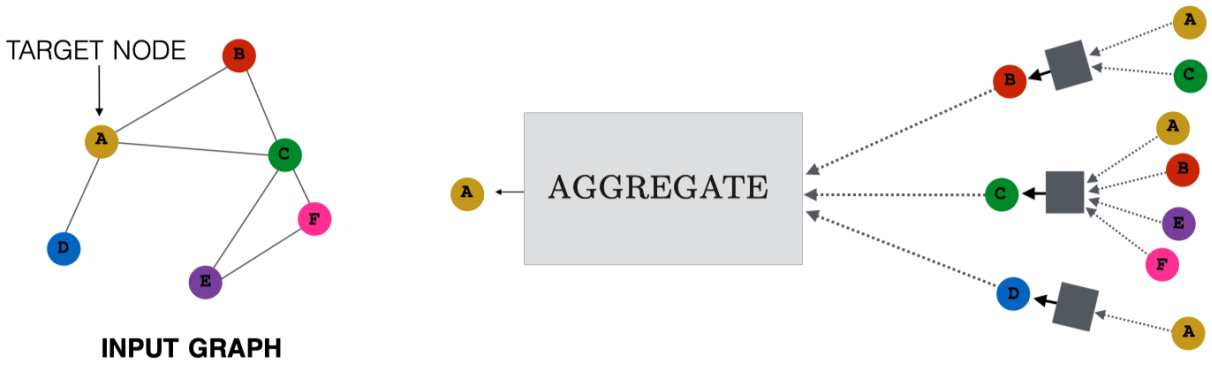

Figure 1: Overview of the message passing framework[9]

$$\mathbf{h}_u^{(l)} = \textbf{UPDATE}^{(l-1)}\left(\textbf{AGGREGATE}^{(l-1)}(\{\mathbf{h}_v^{(l-1)}, \forall v \in N(u)\} \cup \mathbf{h}_u^{(l-1)})\right), \tag{1}$$

where $\mathbf{h}_u^{(l)}$ is the representation of node u in the l-th layer, $N(u)$ is the immediate neighbours of node u. The following models are based on this framework. They apply different aggregation and update functions, in an attempt to achieve better performance.

GCN - Graph Convolutional neural Network This algorithm takes in a graph as a set of feature vectors, and an adjacency structure (matrix, or list). From the inputs, it runs a convolution over the features to produce an updated feature matrix, which is then passed along for further processing[3].

GAT - Graph ATention network These are pretty straightforward if you understand the mechanics behind both GCNs and the transformer architecture. It functions by applying an attention mechanism to the nodes in the same neighbourhood[4].

GraphSAGE - Graph SAmple and aggreGatE GraphSAGE proposes the use of a permutation invariant function to aggregate the information from a node's neighbourhood. This operation allows the network to perform identically among isomorphic graphs[5].

GIN - Graph Isomorphism Network Graph Isomorphism Networks use a multi-layer perceptron to aggregate the information from each node's neighbourhood in the hopes of finding an optimal aggregation function, based on the universal approximation theorem[6].

3.1.2 Proposed Architecture Amendments

Although GNNs have a powerful representation ability, previous studies have shown that the expressive power of existing GNNs is upper-bounded by the 1-Weisfeiler-Lehman (1-WL) test[10]. In the node classification example of Figure 2, v_1 and v_2 cannot be accurately distinguished because the computational graphs they produce are too similar for existing GNNs. To solve this problem, this paper proposes two different architecture amendments. These amendments can be used to augment any of the architectures listed in 3.1.1 to bring these models beyond the upper bound of the 1-Weisfeiler-Lehman (1-WL) test and give them greater expressive power.

ID-GNN Identity information is injected into nodes and used in the message passing stage of the GNN. Only a portion of the nodes are coloured, but because the central node is coloured with a unique colouring, the action is permutation invariant[1]. The first step in processing an ID-GNN is to "colour" a node based on its neighbourhood. We then perform K-rounds of message passing and aggregation before moving on to the next stage. At the second stage of message passing, we use conditional message passing functions, where "coloured" nodes are passed using one function, and "uncoloured" nodes are passed using a different function.

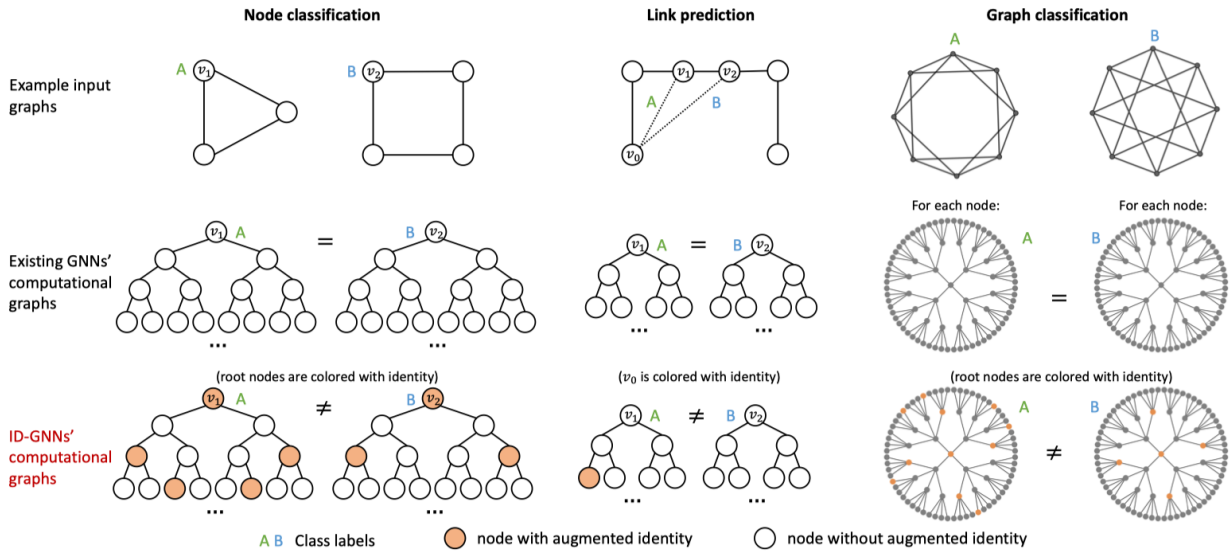

Figure 2: A graphical demonstration of how loops will be seen for different graph structures[1]

Table 1: Statistics of the datasets used in our experiments

Dataset	#Nodes	#Edges	#Graphs	#Features	#Node classes	#Graph classes	Type
Cora	2708	5429	1	1433	7	-	Real-world
Citeseer	3327	4732	1	3703	6	-	Real-world
ENZYMES	19580	37282	600	3	-	6	Real-world
PROTEINS	43471	81044	1113	3	-	2	Real-world
BZR	14479	15535	405	53	-	2	Real-world
ScaleFree	6400	12400	100	-	-	-	Synthetic
SmallWorld	6400	12800	100	-	-	-	Synthetic
ScaleFree500	32000	62000	500	-	-	-	Synthetic
SmallWorld500	32000	64000	500	-	-	-	Synthetic

For edge-level tasks in link prediction, we colour the non-central node instead of the central node. If it is within the computational graph of the central node, we will begin to see messages passed from it propagate.

Figure 2 demonstrates how this graph colouring will be propagated through the local neighbourhood. The message passing for the triangle graph can be seen to reach back to the first node at message passing steps 2 and 3, while with the square node, the message only reaches back at step 2. This is because the other message at step two goes to the lower-right node, which is not connected to the source node, while in the triangle graph, the messages are exchanged across nodes that are also connected to the source node.

ID-GNN Fast Cycle counts are added as an augmented node feature to generate identity information[1]. Cycle counts take a k-hop map of the neighbourhood around a node and use that network to determine how many cycles are present. This augmented feature is concatenated along the node feature axis. This augmentation is simple, adding very little computation to the network.

3.2 Datasets

We utilize 9 graph datasets to perform experiments. Cora and Citeseer are widely used citation network benchmark datasets where nodes are papers, and edges are the citation relationship between two papers. ENZYMES, PROTEINS, and BZR are 3 protein datasets with 600, 1113, and 405 graphs respectively. ScaleFree and SmallWorld are two synthetic datasets. An overview summary of statistics of these datasets is shown in Table 1.

Table 2: Summary of main hyperparameters

Name	#Layers	Optimizer	Max epochs	Learning rate	#Inner dimension	Train/Val	Batch size
Value	3	Adam	1000	0.01	128	0.8/0.2	128

All of the datasets were split into 80% training and 20% validation. There was no test, or hold-out set. PyTorch Geometric and TensorFlow geometric both handle graphs using a slightly different structure, so we needed to build a separate dataset preprocessor.

3.3 Hyperparameters

Hyperparameters were taken directly from the YAML files, and were identical to those listed in the original paper. We confirmed these against the hyperparameters published in the original paper. Table 2 summarizes the main hyperparameters used in our experiments.

3.4 Experimental setup and code

We first implemented 4 existing GNN models with tf_geometric library in TensorFlow, which are used as baselines in the ID-GNN paper. Then we extended the four GNN models based on the Identity-aware mechanism proposed by this paper to get the 4 ID-GNN models. We also implemented the simplified version of ID-GNN by injecting identity (i.e., augmented node features), which is referred to as the ID-GNN fast model. Finally, we conducted the node classification experiment on these 12 models.

The experiments were defined in text documents using a proprietary format. The content of this file would be the name of a variable to adjust, followed by a list of different values to use in that experiment. This would create an n-dimensional grid of experiments to evaluate. These files would then be processed to create YAML files for each individual experiment. The YAML files would then be executed sequentially and the results would be aggregated into a comprehensive document. Each YAML file would contain a hierarchical representation of information such as the number of features used, the number of layers at each stage, the type of convolution, optimizer parameters, and the dataset and task to execute. For ID-GNN fast, we used the same DeepSnap interface that was used by the original authors.

Performance was evaluated using accuracy from a cross entropy loss function, as is common in classification tasks. The code with our current updates is available in the following GitHub repository: `https://github.com/JBanks/GraphGym`.

3.5 Computational requirements

We were able to train the ID-GNN network for 100 epochs on 8x 3.9GHz cores without a GPU in 7 minutes, while the GNN network, and ID-GNN Fast trained in a matter of seconds. For the actual experimentation, we used a single Nvidia RTX 2080 Ti with 11GB of RAM.

4 Results

The results from our implementation matched the results from the original paper, to a degree, and we were able to confirm all but one of the claims outlined in section 2. Across all of the models and all of the augmentations, our results were a near-match to their results, with many of the same networks achieving the highest score for a dataset with similar levels of augmentation. One glaring difference between our results and theirs was with the ID-GNN implementation of the GAT network on the ScaleFree dataset. The results they posted had an accuracy of 63.8%, while our results were much higher at 98.7%. Their result did not seem to follow the patterns outlined in the rest of their results, where GAT was in the top half of the results for each dataset, while ours was consistent with GAT reporting the highest accuracy on that experiment. This discrepancy actually helped with the research claim that ID-GNNs outperform classical GNNs in the node prediction task.

Their second research claim, stating that they don't need to increase computational complexity, was also proved true. For every model, adding in the augmented features helped the model learn a better representation of the nodes. We know that the computation model did not change, because the architecture used between the classic GNN and the ID-GNN Fast were identical.

Table 3: Results of node classification: predicting clustering coefficient (Results from our reproduction)

		ScaleFree	SmallWorld	Enzymes	Proteins
GNN	GCN	**0.695** $\pm$ 0.01	**0.489** $\pm$ 0.05	0.540 $\pm$ 0.06	0.481 $\pm$ 0.01
	SAGE	0.470 $\pm$ 0.03	0.271 $\pm$ 0.03	**0.574** $\pm$ 0.08	0.491 $\pm$ 0.02
	GAT	0.470 $\pm$ 0.03	0.271 $\pm$ 0.03	0.492 $\pm$ 0.07	0.441 $\pm$ 0.02
	GIN	0.639 $\pm$ 0.01	0.470 $\pm$ 0.04	0.543 $\pm$ 0.06	**0.530** $\pm$ 0.01
ID-GNN Fast	GCN	0.764 $\pm$ 0.00	0.571 $\pm$ 0.05	0.724 $\pm$ 0.05	0.728 $\pm$ 0.01
	SAGE	**0.909** $\pm$ 0.01	**0.982** $\pm$ 0.01	**0.956** $\pm$ 0.03	**0.965** $\pm$ 0.01
	GAT	0.581 $\pm$ 0.02	0.616 $\pm$ 0.04	0.636 $\pm$ 0.05	0.621 $\pm$ 0.02
	GIN	0.687 $\pm$ 0.03	0.709 $\pm$ 0.04	0.663 $\pm$ 0.04	0.640 $\pm$ 0.03
ID-GNN Full	GCN	0.964 $\pm$ 0.01	**0.994** $\pm$ 0.00	0.970 $\pm$ 0.03	0.986 $\pm$ 0.01
	SAGE	0.579 $\pm$ 0.07	0.271 $\pm$ 0.03	0.608 $\pm$ 0.07	0.527 $\pm$ 0.01
	GAT	**0.987** $\pm$ 0.00	0.967 $\pm$ 0.04	**0.981** $\pm$ 0.02	**0.991** $\pm$ 0.00
	GIN	0.660 $\pm$ 0.03	0.503 $\pm$ 0.05	0.521 $\pm$ 0.09	0.540 $\pm$ 0.01
Best ID-GNN over best GNN		29.2%	50.5%	40.7%	46.1%

Table 4: Results of node classification: predicting clustering coefficient (Results from the original paper)

		ScaleFree	SmallWorld	Enzymes	Proteins
GNN	GCN	0.679 $\pm$ 0.01	**0.589** $\pm$ 0.04	0.596 $\pm$ 0.02	0.540 $\pm$ 0.00
	SAGE	0.470 $\pm$ 0.03	0.271 $\pm$ 0.03	0.572 $\pm$ 0.04	0.444 $\pm$ 0.03
	GAT	0.470 $\pm$ 0.03	0.274 $\pm$ 0.06	0.464 $\pm$ 0.03	0.400 $\pm$ 0.02
	GIN	**0.693** $\pm$ 0.00	0.571 $\pm$ 0.04	**0.660** $\pm$ 0.02	**0.558** $\pm$ 0.02
ID-GNN Fast	GCN	0.897 $\pm$ 0.01	0.812 $\pm$ 0.02	0.786 $\pm$ 0.04	0.805 $\pm$ 0.02
	SAGE	**0.954** $\pm$ 0.01	**0.994** $\pm$ 0.00	**0.958** $\pm$ 0.04	**0.985** $\pm$ 0.01
	GAT	0.889 $\pm$ 0.01	0.739 $\pm$ 0.03	0.675 $\pm$ 0.04	0.675 $\pm$ 0.05
	GIN	0.895 $\pm$ 0.00	0.822 $\pm$ 0.03	0.798 $\pm$ 0.06	0.790 $\pm$ 0.01
ID-GNN Full	GCN	**0.985** $\pm$ 0.01	**0.994** $\pm$ 0.00	0.984 $\pm$ 0.02	**0.995** $\pm$ 0.00
	SAGE	0.588 $\pm$ 0.01	0.400 $\pm$ 0.12	0.591 $\pm$ 0.04	0.474 $\pm$ 0.03
	GAT	0.638 $\pm$ 0.01	0.847 $\pm$ 0.20	**0.994** $\pm$ 0.00	0.994 $\pm$ 0.00
	GIN	0.716 $\pm$ 0.01	0.572 $\pm$ 0.04	0.655 $\pm$ 0.02	0.570 $\pm$ 0.03
Best ID-GNN over best GNN		29.3%	40.6%	33.4%	43.7%

The average improvement across datasets for our reproduction was actually higher than their model with 41.6% with synthetic datasets, and 32.7% overall. Their improvements were only 36.8% on synthetic datasets, with 24.9% overall. Our improvement was highly influenced by a (likely erroneous) accuracy of 100% with GraphSAGE and GAT on the Cora dataset, and a (likely erroneous) accuracy of $\tilde{9}5\%$ with the same models on the CiteSeer dataset. This erroneous result creeped in during the final day before the report was due, and we were unable to locate the bug.

The values achieved by the baseline models on the real datasets were so high, and the improvements on the synthetic datasets were so good, that claim 3 must hold, even when accounting for the erroneous results. Even with removing the erroneous results, we are still able to show that claim 5 holds. If we remove the results that we assume to be erroneous, we are not able to show a 1% improvement on the real datasets, so claim 4 does not hold.

5 Discussion

We believe that the model we built can be refined to produce improved results and we will continue work on improving it. One thing that we did differently was to simplify the overall architecture. There may be some nuance in the complicated architecture that they built to achieve their impressive results.

Table 5: Results of node classification: real-world labels (Results from our reproduction)

		Cora	CiteSeer
GNN	GCN	**0.879** ± 0.00	0.763 ± 0.01
	SAGE	**0.879** ± 0.00	0.762 ± 0.02
	GAT	0.878 ± 0.00	**0.770** ± 0.01
	GIN	0.835 ± 0.01	0.702 ± 0.02
ID-GNNs Fast	GCN	0.880 ± 0.01	0.756 ± 0.01
	SAGE	0.878 ± 0.01	0.754 ± 0.01
	GAT	**0.881** ± 0.01	**0.759** ± 0.00
	GIN	0.809 ± 0.05	0.678 ± 0.01
ID-GNNs Full	GCN	0.787 ± 0.03	0.767 ± 0.00
	SAGE	**1.000** ± 0.00	0.938 ± 0.09
	GAT	0.885 ± 0.00	0.771 ± 0.01
	GIN	**1.000** ± 0.00	**0.948** ± 0.07
Best ID-GNN over best GNN		12.1%	17.8%

Table 6: Results of node classification: real-world labels (Results from the original paper)

		Cora	CiteSeer
GNN	GCN	0.848 ± 0.01	0.709 ± 0.01
	SAGE	**0.868** ± 0.01	**0.726** ± 0.01
	GAT	0.857 ± 0.01	0.716 ± 0.01
	GIN	0.858 ± 0.01	0.719 ± 0.01
ID-GNNs Fast	GCN	0.851 ± 0.02	0.715 ± 0.00
	SAGE	0.866 ± 0.02	**0.742** ± 0.01
	GAT	**0.870** ± 0.02	0.719 ± 0.02
	GIN	0.864 ± 0.01	0.719 ± 0.01
ID-GNNs Full	GCN	0.863 ± 0.01	0.719 ± 0.01
	SAGE	0.875 ± 0.01	**0.730** ± 0.02
	GAT	**0.878** ± 0.01	0.729 ± 0.01
	GIN	0.851 ± 0.00	0.725 ± 0.01
Best ID-GNN over best GNN		1.0%	1.6%

5.1 What was easy

The requirements for running their code are fully outlined in the README.md file on GraphGym's GitHub page. Our initial attempt to install and execute GraphGym was outside of a virtual environment. There were some problems with dependencies that we did not dig further into. Running a Conda virtual environment and following the exact steps outlined in the readme allowed us to get an environment capable of executing the author's original code.

Implementing the ID-GNN Fast architecture was relatively easy compared to the full ID-GNN architecture. It only required including an extra set of features that were concatenated onto the end of the existing vector.

5.2 What was difficult

There was a lot of background knowledge required that someone unfamiliar with GNNs would not have.

There were a number of issues surrounding code comprehension when first stepping into the code base. They had developed a proprietary system for running their experiments, which allowed them to quickly iterate through experiments, but added to complexity of knowing exactly what was required for each experiment. Certain experiments required the data to be transformed in different ways, or required the specification of whether the graph was transductive. The YAML files that were pre-built with the software package did not actually outline these nuances in the datasets, and were merely used as templates for generating the YAML files based on a list of properties from a 'txt' document.

These misunderstandings caused a lot of headaches while we worked on setting up the datasets and running the initial experiments.

When trying to understand the flow of their code, portions of the code base used different python mechanisms to accomplish the same task. This was done in a way that made the code very easy to extend if you were implementing a new module in PyTorch, but made it difficult to trace back through the code. They made heavy use of global variables and dictionaries that had entries generated within multiple different python scripts using different pythonic mechanisms for adding these items (e.g. appending to the list by using the ** operator in some instances, and using a "register" function in others). Eventually we decided against implementing our code alongside theirs because we were spending too much time tracing code to add in the compatibilities, and not enough time coding our implementations.

When attempting to run their experiments as they had originally coded the solution, we ran into some issues. This made it so that we could not compare their published results with the results that they advertised. More specifically, we received the following error: RuntimeError: The expanded size of the tensor (1) must match the existing size (376932) at non-singleton dimension 1. Target sizes: [376932, 1, 128]. Tensor sizes: [1, 376932, 1]. We did not investigate why this was happening, and instead proceeded with our own implementation.

We experienced a lot of trouble with version compatibility between software required to run the experiments, as well as version compatibility within the same software package. These package incompatibilities led us to reproducing some of the features from these libraries so that they would work regardless of the version that the end user has installed. Some software would not work on the native operating system and needed to be installed via a Conda virtual environment.

For some dataset-model pairings we were achieving results with 100% accuracy. This was obviously incorrect, and we were unable to determine where the problem was in the code. We had the same system across all models and datasets, so only seeing these results on a small number of pairings was surprising.

5.3 What was learned

After working through their research, we became familiar with several GNN models and how they work. This will be invaluable knowledge as we proceed onto further research using unstructured data that is represented as a set of graphs.

We also familiarized ourselves with how to alter the structure of our input to perform feature engineering and achieve better results from existing models. This may provide avenues of improving performance of models without going through complex architectures or spending hours fine-tuning hyperparameters.

The framework for GraphGym is now a standard framework which will be valuable to be familiar with. Having this knowledge going forward will allow us to easily plug in new modules and datasets to quickly determine which direction to search in, or to quickly develop prototypes from which to fine tune.

We also learned a lot from seeing the coding style of the other person. Each person brought a different skillset and perspective to the code, which allowed us to take the best parts of our knowledge bases and produce a better overall result. We will be able to use the tricks learned to improve our code for future projects.

5.4 Communication with original authors

We have not yet reached out to the authors. We felt that we produced a fair representation of their work, and will reach out to them only if we decide to proceed with the reproducibility publication.

References

[1] J. You, J. M. Gomes-Selman, R. Ying, and J. Leskovec, "Identity-aware graph neural networks," 2021. [Online]. Available: http://snap.stanford.edu/idgnn

[2] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[3] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2017.

[4] P. Veličković, A. Casanova, P. Liò, G. Cucurull, A. Romero, and Y. Bengio, "Graph attention networks," 2018.

[5] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," vol. 2017-December, 2017.

[6] K. Xu, S. Jegelka, W. Hu, and J. Leskovec, "How powerful are graph neural networks?" 2019.

[7] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner *et al.*, "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:1806.01261*, 2018.

[8] J. Hu, S. Qian, Q. Fang, Y. Wang, Q. Zhao, H. Zhang, and C. Xu, "Efficient graph deep learning in tensorflow with tf_geometric," 2021.

[9] W. L. Hamilton, "Graph representation learning," *Synthesis Lectures on Artifical Intelligence and Machine Learning*, vol. 14, no. 3, pp. 1–159, 2020.

[10] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" 2019.