Reproducibility report for Identity-Aware Graph Neural Networks

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

Zhihao Dong
School of Computing
Queen's University
Kingston, ON
z.dong@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 40% accuracy improvement on node, edge, and graph structure (i.e., clustering coefficient) prediction tasks by transforming it with the ID-GNN extension. The improvement becomes smaller on real-world dataset benchmarks, only 3% on node and graph classification and 15% on link prediction 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 the results seen in the author's paper.

Results

Through our partial implementation, We have found no improvements in ID-GNN-Fast. The SmallWorld dataset using the GAT architecture reported an improvement from 27% to 73%, while ours was stable at 30%.

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. With the code accessibility and runability, we do not foresee this being a step that we will need to take.

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

1 Introduction

Graph Neural Networks (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. ID-GNNs are an attempt to create a generalized solution to apply in any graph scenario.[1] The paper we are reproducing 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 heterogeneous 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 40% 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[2]. This is a library for graph-related neural network processing. It is similar to PyTorch Geometric, which was used by the authors of the original paper.

While training our model, we used a single Nvidia RTX 2080 Ti with 11GB of RAM.

3.1 Model descriptions

3.1.1 Classical GNN Architectures

The following is a list of the most common GNN architectures used in Machine Learning:

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 a 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]

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

3.1.2 Proposed Architecture Amendments

This paper proposes two different architecture amendments. These amendments can be used to augment any of the architectures listed in 3.1.1.

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.

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.

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, 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.

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.

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.

3.4 Experimental setup and code

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

		ScaleFree	SmallWorld	Enzymes	Proteins
GNN	GCN	0.570833385	0.461197942	0.451421499	0.376320124
	GAT	0.470312506	0.303385407	0.4522686	0.358956963
ID-GNN Fast	GCN	0.654947937	0.628385365	0.655619442	0.652693093
	GAT	0.577343762	0.594270825	0.615869462	0.61372304

Table 2: Results of node classification: predicting clustering coefficient

		Cora	CiteSeer
GNN	GCN	0.883148849	0.771271288
	GAT	0.88437885	0.763263285
ID-GNN Fast	GCN	0.891143858	0.769269288
	GAT	0.886223853	0.771271288

Table 3: Results of node classification: real-world labels

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

Our preliminary results are not in line with the claims of the paper. Table 2 shows that our results for clustering coefficients had no improvement for the ID-GNN strategy. Table 3 again, shows a lack of improvement when using the ID-GNN architecture. These results do not reflect the claims of the paper.

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.

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 were a number of issues around familiarizing ourselves with the platform:

- There were a number of hyperparameter YAML files created in the configs directory, but they were a red herring, and the actual hyperparameter files were generated from the txt files in the grids directory

- Shell scripts reference "python", which can cause issues on systems that have both python, and python3 installed.
- Certain libraries (torch_sparse) had issues installing when we attempted to run the original code. We were able to get around it by using a Conda environment
- The platform uses a lot of dictionary calls with configuration options being the keys. PyCharm did not allow us to Ctrl+Click to quickly navigate through the code, and we had to constantly reference the YAML file, the dictionary creation, and the specific file holding the function we were referencing.
- When initially looking at the dictionary establishment, we were looking for the "register" function which is used in some parts of the code base. Other parts of the code base use dictionary expansion with double asterisk (**) to initialize the dictionary with values.
- We received the following error when attempting to run the original code across a number of configurations. We did not investigate how to solve it, or what the root cause was.

  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]
- In the paper, they talk about SmallWorld and ScaleFree datasets that they created with 256 graphs. We were able to find datasets with 100 and 500 graphs referenced in their code's hyperparameter search, but not the sets with 256.

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

5.3 Communication with original authors

We have not yet reached out to the authors. We would like to have initial results from our experimentation before we begin posing questions.

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] 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.

[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.