ML Reproducibility Challenge 2021: Beyond Low-frequency Information in Graph Convolutional Networks

Fabian Beringer
Leibniz Universität Hannover
f.beringer@stud.uni-hannover.de

Reproducibility Summary

Scope of Reproducibility

In this work we try to reproduce results from "Beyond Low-frequency Information in Graph Convolutional Networks" Bo et al. [2021]. The authors claim their proposed architecture named Frequency Adaptation Graph Convolutional Network (FAGCN) achieves higher accuracy than previous GNN architectures, when performing node classification on both assortative and in particular disassortative graphs. We re-implement FAGCN, train it on the same datasets as in the paper and compare its performance to a standard GCN as baseline.

Methodology

We base our re-implemention of FAGCN on the paper's model description and train on multiple datasets, according to the author's experiments. For this we make use of existing Pytorch GeometricFey and Lenssen [2019] modules and infer missing details from the author's code.

Results

Through our experiments, we did not observe a significant improvement of FAGCN over the baseline when training on assortative networks, which does not support the author's claim of FAGCN outperforming GCN on assortative networks. However, for disassortative networks we did measure significant improvements in accuracy compared to GCN, supporting the authors claim that FAGCN is better at handling disassortative networks than GCN approaches.

What was easy

1. A concise and simple description of the model facilitated re-implementation.
2. Detailed description of all hyperparameters enabled a rather exact reproduction of the experiments.

What was difficult

1. Some architecture details were only inferrable from code, namely position of dropout layers and type of activation function.
2. The training splits for disassortative networks were not provided, preventing exact comparisons.

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

1 Introduction

Recently, neural networks have become a popular approach for solving tasks that require operating on data represented as graph structures. One of these tasks is node classification where, given a graph, the goal is to predict the class of a node. Graph Neural Networks (GNNs) can leverage structure of a graph in order to increase predictive ability. This is achieved by recursively collecting and aggregating information from a node's neighbourhood, to generate a richer feature representation of the node itself.
There are numerous different variations in GNN architecture and exploring new architectures, as well as improving upon existing ones is an active field of research.
With the emergence of more and more novel GNN approaches, the verification of their effectiveness also becomes an important task. For this work, we take a look at the newly proposed architecture named FAGCN and try to verify the findings of its associated paper.

2 Scope of reproducibility

This work tries to replicate the findings from "Beyond Low-frequency Information in Graph Convolutional Networks" Bo et al. [2021]. In their paper the authors propose a new graph neural network (GNN) architecture called "Frequency Adaptation Graph Convolutional Network" (FAGCN). The architecture is designed around the idea of previous GNN architectures, e.g. GCN Kipf and Welling [2017], mostly leveraging low-frequency neighbour information in order to learn new node representations.
However, the authors argue that for node classification, high-frequency information can also be important, especially when dealing with disassortative networks, i.e. graphs in which dissimilar nodes tend to group together.
In order to adaptively include both high- and low-frequency signals, they propose an attention based gating mechanism which automatically controls the amount of low- and high-frequency neighbour information that influences a nodes representation.
To verify the benefits of this approach for disassortative networks, FAGCN is trained on the Chameleon, Squirrel Rozemberczki et al. [2019] and Actor Tang et al. [2009] datasets with a node classification objective. Further, common assortative datasets, namely Cora, Citeseer and Pubmed Yang et al. [2016] are used to test the model's ability to adapt between assortative and disassortative networks.
In order to evaluate the performance of the FAGCN model, we focus on verifying the following claims:

1. FAGCN achieves accuracy higher but close to GCN on cora, citeseer and pubmed assortative networks.
2. FAGCN achieves higher accuracy on dissasortative datasets, which GCN fails to adapt to.

Note that for simplicity we will focus on the GCN architecture for direct comparison. While the authors of the original paper also compare accuracy to a number of other different models, the GCN serves as bottom baseline and will be sufficient for our purposes. Furthermore, while in the original paper additional experiments were made using synthetic datasets, we limit our scope to the previously mentioned real world benchmark datasets.

3 Methodology

We conduct our experiments using PyTorch Paszke et al. [2019] in combination with the PyTorch Lightning framework Falcon et al. [2019]. For the model implementation we also make use of the PyTorch Geometric Fey and Lenssen [2019] library for graph neural networks.
We re-implement the FAGCN model as described in the paper, using the in-build "Frequency Adaptive Graph Convolution" operation (namely FAConv). The operation implements a single step of neighbour aggregation and update of the of hidden node representations for FAGCN.
Since some implementation details were missing in the paper, we also look into the authors original implementation found on github[1] which contained the missig details.
We run our experiments on a single desktop machine equipped with a Ryzen 3700x 8-core processor, an RTX 3090 GPU and 32GB of RAM. All package dependencies and versions that were used can be found in our repository.

[1] `https://github.com/bdy9527/FAGCN`

3.1 Model descriptions

We re-implement two types of GNN models:

1. Frequency Adaptation Graph Convolutional Network.
2. Graph Convolutional Network.

FAGCN can be described through the following set of equations:

$$\begin{aligned}
\mathbf{h}_i^{(0)} &= \text{ReLU}(\mathbf{W}_{in}\mathbf{x}_i + \mathbf{b}_{in}) \\
\mathbf{h}_i^{(l)} &= \varepsilon\mathbf{h}_i^{(0)} + \sum_{j\in N(i)} \frac{\alpha_{ij}}{\sqrt{d_i d_j}}\mathbf{h}_j^{(l-1)} \\
\mathbf{h}_{out,i} &= \mathbf{W}_{out}\mathbf{h}_i^{(L)} + \mathbf{b}_{out} \\
\\
\alpha_{ij} &= \tanh(\mathbf{g}^\top[\mathbf{h}_i;\ \mathbf{h}_j])
\end{aligned} \tag{1}$$

Where $\mathbf{W}_{in}, \mathbf{W}_{out}, \mathbf{b}_{in}, \mathbf{b}_{out}$ are learned weight matrices and corresponding bias vectors, $N(i)$ is the set of neighbours of node i, d_i its degree, $\mathbf{x}_i$ the node's initial representation and $\mathbf{h}_i^{(l)}$ its hidden representation at layer $l \in (0, 1, ..., L)$. Further α_{ij} are attention weights computed by concatenating the hidden representation of node i and its first order neighbour j (concatenation denoted by ;) and projecting them using the learned weight vector $\mathbf{g}$. The hyperparameter $\varepsilon \in [0, 1]$ controls how much of the initial node representation to keep during each update.

GCN can be described as follows:

$$\begin{aligned}
\mathbf{h}_i^{(0)} &= \mathbf{x}_i \\
\mathbf{h}_i^{(l)} &= \text{ReLU}\left(\mathbf{W}^{(l)} \sum_{j\in N(i)\cup i} \frac{\mathbf{h}_i^{(l-1)}}{\sqrt{d_i d_j}} + \mathbf{b}^{(l)}\right) \\
\mathbf{h}_{out,i} &= \mathbf{W}_{out}\mathbf{h}_i^{(L)} + \mathbf{b}_{out}
\end{aligned} \tag{2}$$

where the same naming conventions apply as for FAGCN. We train all models from scratch using the Adam Kingma and Ba [2015] optimization algorithm.

3.2 Datasets

For Cora, Citeseer and Pubmed we use the public train-splits as provided in the PyTorch Geometric library.
As there is no public split for the Chameleon, Squirrel and Actor datasets and the authors did not provide the exact splits used in their experiments, following the authors we split each of them into 60% train, 20% validation and 20% test. While this is not ideal for reproducing the exact numbers, we can still compare FAGCN relative to GCN on our custom training splits. The exact splits are provided in our git repository.
Since the raw Chameleon, Squirrel and Actor datasets are designed for regression, we also use pre-binned labels (and also training instances) provided from the authors repository, mapping the regression targets into 3 different classes. Further, each graph is converted to a bidirectional graph.
Dataset statistics are presented in Table 1.

Dataset	Nodes	Edges	Features	Classes
Cora	2,708	10,556	1,433	7
Citeseer	3,327	9,104	3,703	6
Pubmed	19,717	88,648	500	3
Chameleon	2,277	36,101	3,132	3
Squirrel	5,201	217,073	3,148	3
Actor	7,600	30,019	932	5

Table 1: Summary of dataset statistics. The number of edges is reported after conversion to the bidirectional graph used for training.

3.3 Hyperparameters

For FAGCN we follow the exact settings as described in the original paper for each dataset. Namely we set $\varepsilon = 0.2, 0.3, 0.3$ for Cora, Citeseer and Pubmed respectively. Additionally we set hidden size to 16, number of hidden layers $L = 4$ and Adam weight decay to 1e-3. We also apply dropout with rate 0.6 at the input layer and at each hidden layer (not including the output layer).
The hyperparameter settings for Chameleon, Squirrel and Actor are: $\varepsilon = 0.4, 0.3, 0.5$ respectively, hidden size 32, $L = 2$, dropout rate 0.5 and weight decay 5e-5. For all datasets, the learning rate is set to 1e-2.
For GCN we apply the same settings as for FAGCN, however we fix the number of layers to 2, since GCN is prone to oversmoothing when $L > 2$. We do not perform hyperparameter search for GCN as these settings already lead to performance comparable to those reported in the original paper.

3.4 Experimental setup and code

While FAGCN's weights are initialized using Glorot initialization Glorot and Bengio [2010] (as in the authors code), for GCN we use PyTorch's default initialization.
We use a standard train-, validation- and test-split setup. Each model is trained for up to 500 epochs, with early stopping after 200 epochs if there is no improvement in validation set loss. As in the original paper, for each dataset we perform 10 training runs and compute their mean accuracy, as well as the standard deviation.
Our code can be found on github[2]. The repository includes model and training implementation, as well as data pre-processing and the resulting training splits.
Additionally, we provide a report with training curves for each run and the corresponding random seeds used.

3.5 Computational requirements

We train on a single desktop machine equipped with a Ryzen 3700x 8-core processor, an RTX 3090 GPU and 32GB of RAM. On this hardware, both models take from 10-30 seconds for a single training run, depending on the dataset and use <4GB of GPU Memory.

4 Results

We summarize mean and standard deviation of test accuracy over 10 runs in Table 2. We find that for the assortative networks Cora and Pubmed, FAGCN performs similar to GCN, with GCN taking a slight lead. However, for Citeseer, GCN achieves higher accuracy than FAGCN by a 5% margin.
On the disassortative graphs, FAGCN consistently outperforms GCN. It achieves up to >10% higher test set accuracy than GCN.
While the performance on the disassortative datasets matches the original paper's reports, for assortative our results differ, namely FAGCN does not achieve higher accuracy than GCN.

4.1 Results reproducing original paper

4.1.1 Result 1: Assortative Networks

To verify whether FAGCN achieves higher accuracy than GCN on the assortative benchmarks, we re-train GCN and FAGCN on Cora, Citeseer and Pubmed. From Table 2 we can see that while FAGCN achieves comparable performance

[2] `https://github.com/yolomeus/ml4g-project`

	GCN	**FAGCN**	**FAGCN (original)**
Cora	$\mathbf{81.17}\% \pm 0.74\%$	$80.94\% \pm 1.38\%$	$84.1\% \pm 0.5\%$
Citeseer	$\mathbf{67.74}\% \pm 1.33\%$	$62.13\% \pm 3.08\%$	$72.7\% \pm 0.8\%$
Pubmed	$\mathbf{78.89}\% \pm 0.59\%$	$78.63\% \pm 0.52\%$	$79.4\% \pm 0.3\%$
Chameleon	$64.26\% \pm 1.70\%$	$\mathbf{75.21}\% \pm 1.31\%$	not applicable
Squirrel	$52.22\% \pm 1.93\%$	$\mathbf{63.90}\% \pm 1.02\%$	not applicable
Actor	$28.01\% \pm 0.50\%$	$\mathbf{35.19}\% \pm 0.84\%$	not applicable

Table 2: Average test accuracy and standard deviation over 10 runs on each dataset. Higher score between GCN and FAGCN in bold. Disassortative networks are not comparable to the original report as the original splits were not provided.

to GCN on Cora and Pubmed, on Citeseer it is outperformed by a large margin. This result is not consistent with the authors experiments, where FAGCN achieves higher accuracy than GCN on all three datasets.
Further, when training with the specified hyperparameters, for citeseer we also observe high variance of the training loss over runs (See Figure 1). Even though this effect is less pronounced for the other datasets, we can still clearly observe a difference in variance to GCN[3]. This suggests that FAGCN is less stable during training which might explain the differences in observations.

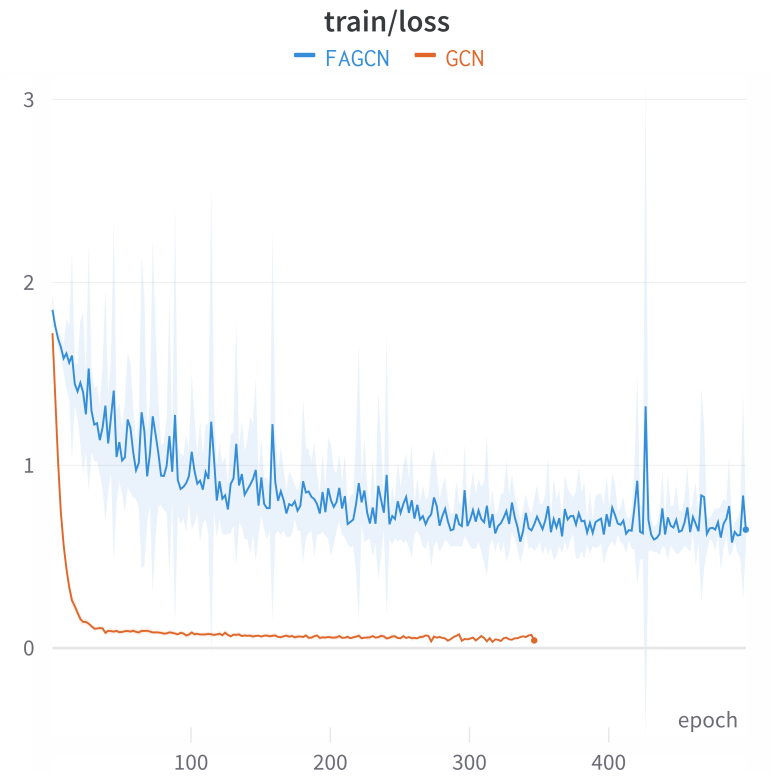

Figure 1: Training loss of FAGCN and GCN during training on Citeseer. FAGCN shows much higher variance.

4.1.2 Result 2: Disassortative Networks

As our splits are not the exact same as the original author's, we can not directly compare scores. However, we can measure the relative difference in performance to our baseline. With that in mind, our findings for the disassortative networks do match the authors findings. On all three datasets, FAGCN outperforms GCN. On Chameleon and Squirrel, FAGCN achieves $\sim 11\%$ and on Actor $\sim 7\%$ higher accuracy. However, the margins in the original report are smaller: $\sim 5\%$ difference were reported for chameleon and squirrel and $\sim 6\%$ for actor.

5 Discussion

Our experimental results can only partially support the authors claims. While we were able to verify that FAGCN has an advantage on the disassortative graphs presented in the paper, we were not able to replicate the same performance on the assortative networks. Possible reasons for this mismatch could have been:

- The public datasets differ in some way from the author's
- Differences between PyTorch Geometric and the original implementation
- Different convergence properties on our hardware, given the same hyperparameters

[3]Detailed training histories can be found at:
`https://wandb.ai/yolomeus/ml4g-project/reports/ML4G-Project--VmlldzoxNDU5ODQz`

For further investigation, it might be necessary to perform extensive hyperparameter search for both FAGCN and GCN to get a more fair comparison which is beyond the scope of this work.

5.1 What was easy

The explanation of the model architecture was detailed enough for re-implementation. Also, the authors reported all hyperparameters used for each dataset and details on how training was performed. The code provided was concise and readable.

5.2 What was difficult

While most of the model architecture was described in the paper, two details were missing:

1. The position at which layers dropout was performed.
2. Which activation function ϕ was used for the input transformation.

However, these details could easily be derived from the authors implementation. Unfortunately, there was no documentation for the source code at all.
Furthermore, while the ratio of training splits for the disassortative networks was mentioned in the paper, the authors did not provide the exact splits for their experiment. This made comparisons to the original report harder, since we had to train on our own splits.

References

Deyu Bo, Xiao Wang, Chuan Shi, and Huawei Shen. Beyond low-frequency information in graph convolutional networks. *ArXiv*, abs/2101.00797, 2021.

Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding. *CoRR*, abs/1909.13021, 2019. URL `http://arxiv.org/abs/1909.13021`.

Jie Tang, Jimeng Sun, Chi Wang, and Zi Yang. Social influence analysis in large-scale networks. In *KDD*, 2009.

Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. *CoRR*, abs/1603.08861, 2016. URL `http://arxiv.org/abs/1603.08861`.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

William Falcon et al. Pytorch lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, 3, 2019.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and D. Mike Titterington, editors, *AISTATS*, volume 9 of *JMLR Proceedings*, pages 249–256. JMLR.org, 2010.