Per-node-depth GNN: using centrality measures to classify optimal depth

Rotem Ben Zion
rotm@campus.technion.ac.il

Niv Kook
niv.ko@cs.technion.ac.il

Abstract

Graph Neural Networks (GNNs) are neural networks designed to process data in graph form. Unlike other types of neural networks, empirical experiments show that the performance of GNNs drops significantly with large depths. Therefore, SOTA GNNs are shallow networks with few layers only.

In this project, we explore the idea of per-node-depth; the final representation of each node can be calculated by an individual amount of message-passing steps. More specifically, instead of a single classifier using the final (full-depth) representation, we train a node-classification network for each level of our GNN, and evaluate the optimal depth per node as a function of pre-calculated centrality measures. We analyze our results and deduct on the contribution of different node characteristics to its optimal depth.

1 Introduction

Motivated by the success of Deep Learning (DL), several attempts have been made to apply DL models to non-Euclidean data; particularly, graph-structured data such as chemical compounds, social networks, and polygons. The idea of Graph Neural Networks (GNNs) was proposed in (Gori et al., 2005) and (Scarselli et al., 2008), which incited many research papers over the last two decades.

one of the major problems in training GNNs is their struggle to propagate information between distant nodes in the graph (Alon and Yahav, 2020). The distance between nodes is synonymous with the depth of the network, as the mechanism of GNNs operates by message-passing, which means the receptive field of a given node is simply the depth of the network in hop-distance. As a result, state-of-the-art GNN models are mostly shallow networks, in sharp contrast to the best performing models in other areas of DL such as Computer Vision and NLP. As will be discussed in section 2 (related work), many papers presented potential causes, analysis and solutions to this phenomenon. However, most of these works implicitly assumed that depth should be constant over the graph, by inputting the final node embedding to a single trained classifier.

In this project, we deviate from this convention, and apply per-node-depth; we insert a classifier after each message-passing layer, so that the node classification task can be done using different depths for different nodes. Our end goal is to investigate the question: **can the optimal depth of a given node be derived from its centrality measures in the graph?**

To do so, we build a dataset by extracting centrality measures as features for each node, and assigning labels based on a comparison between the classifiers, as explained in section 3 and algorithm 2. We then test the relationship between the features and labels in this new dataset by statistical analysis and by training ML models and testing their generalization and performance. Our hypothesis is that centrality measures should contain enough information to reasonably deduct the optimal depth. The justification for our hypothesis is that the well-established obstacle in GNN training - oversmoothing - directly relates to the amount of information aggregated by a single node, which in turn is intuitively correlated with the node's centrality.

2 Related work

Many works in the past few years revolved around the exploration of depth in GNNs.

The article "Do we need deep graph neural networks?" by Michael Bronstein (tow) discusses motivation for deep GNNs as well as possible explanations to the difference from networks in computer vision or NLP. (Oono and Suzuki, 2019) show the decrease in expressive power of GCNs (Graph Con-

volutional Network) as depth increases. (Alon and Yahav, 2020) propose an explanation to this phenomenon, claiming that GNNs are susceptible to a bottleneck when aggregating messages across a long path, causing *over-squashing* of information. The authors in (Li et al., 2018) suggest that the main difficulty in constructing deep networks is lying in over-smoothing. (Rong et al., 2019) attempt to deal with overfitting and over-smoothing by randomly dropping graph edges during each training epoch, and the authors of (Li et al., 2019) use residual/dense connections and dilated convolutions to train a 56-layer GCN with good performance. More related to our project is the paper "RED-GCN: Revisit the Depth of Graph Convolutional Network", which suggests a method to find the optimal depth d of a given graph without any prior knowledge and without the constraint to positive integers. However, the depth is still on a graph-level and not node-level. For the tasks of node classification and edge classification, the paper (Zeng et al., 2021) suggests a method which does allow individual depths, by first extracting localized subgraphs, and then applying GNNs with arbitrary depths on top of the subgraphs. Unfortunately, the extraction of subgraphs requires identifying the influential neighbors of each entity which is generally difficult.
Lastly, on the topic of centrality measures, we consulted this Wikipedia page.

3 Our method

As previously mentioned, our goal is to test whether node centrality measures contain sufficient information to determine the optimal depth of a given node. This entails a few challenges:

- How can we evaluate, over the nodes in our training set, which depth is optimal?
- How can we capture the relationship between the extracted features and the optimal depth?

With these challenges in mind, we propose the following method. Let max_depth be the upper bound on depths we wish to test. The method consists of three main steps.

1. We define a GCN with max_depth message-passing layers. We insert a classifier after each layer, denoted by $\{clf^{(i)}\}_{i=1}^{max\_depth}$. The loss used to train the network will now be the simple summation of losses. The training procedure is specified in alorithm 1.
2. After the network and classifiers are trained, we take a new set of heldout nodes, and for each we compare the different classifiers and extract the best one (loss-wise). This will be defined as the label in our new dataset. Pseudocode for this process presented in algorithm 2.
3. We use our newly created dataset to train a classification model. The model choice is done via cross-validation and can be a decision tree or an MLP. We then use the fitted classifier to test generalization and the influence of different centrality measures as desired.

Centrality measures

In this subsection, we list the centrality measures used in our implementation.

1. **in-degree**: number of edges into the node.
2. **out-degree**: number of edges out of the node.
3. **PageRank**: importance estimatation by counting the number and quality of edges into the node.
4. **Hubs**: part of the HITS algorithm, measures how well the node points to good authorities.
5. **Authorities**: part of the HITS algorithm, measures how many good hubs point to the node.
6. **Katz centrality**: computes the relative influence of a node, by counting neighbors of different hop distances with a discount factor.
7. **Perculation**: attempts to evaluate how "infectious" the node is, by counting the proportion of shotest paths that go through the node.
8. **Eigenvector centrality**: assigns a relative score based on the concept that connections to high-scoring nodes contribute more than equal connections to low-scoring nodes. PageRank and Katz centrality are variants of the eigenvector centrality.

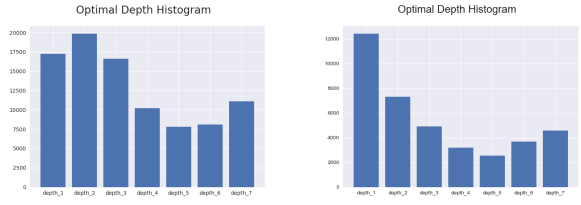
Imortant note: several centrality measures were numerically impossible to compute, time-wise or memory-wise. For example, the betweenness measure. This could potentially be solved by calculating the measures on subgraphs, but as this project's goal is a proof of concept, we have decided to only use the centrality measures stated above.

Algorithm 1 Network Training Procedure

for batch in Training Set **do**
    $X =$ batch.features
    $y =$ batch.labels
    loss $= 0$
    for $i \in \{1, 2, \ldots,$ max_depth$\}$ **do**
        $X = layer^{(i)}(X)$
        pred $= clf^{(i)}(X)$
        loss $=$ loss $+$ cross_entropy(pred, y)
    end for
    $W_{\text{network}} = W_{\text{network}} - \alpha \cdot \nabla$loss
end for

Algorithm 2 Optimal-depth Data Creation

Assume $\{layer^{(i)}\}_{i=1}^{\text{max\_depth}}, \{clf^{(i)}\}_{i=1}^{\text{max\_depth}}$ are trained on the training set
Initialize an empty dataset $D = \emptyset$
for node in a heldout dataset **do**
    $X =$ node.features
    $y =$ node.label
    clf_scores $= \{\}$
    for $i \in \{1, 2, \ldots,$ max_depth$\}$ **do**
        $X = layer^{(i)}(X)$
        pred $= clf^{(i)}(X)$
        clf_scores$[i]$ $=$ cross_entropy(pred, y)
    end for
    depth_label $= \arg\min$ clf_scores
    centrality_features $=$ precalculated_cent_features(node)
    Add data point (centrality_features, depth_label) to D
end for
Return D



(a) train set (b) heldout set

Figure 1: Histograms of optimal depths

4 Experiments

In order to test our hypothesis, we have implemented the following set of experiments.

- Optimal depth histogram, to cehck class imbalance in our new dataset
- Performance and generalization of the depth classifier with different max_depth values
- Correlation heatmap of the different centrality measures with the optimal depth

The dataset used in our implementation is the obgn-arxiv dataset (Hu et al., 2020).
All code, including network training, depth classifier, data processing and experiments is available on GitHub. All of the experiments were run on Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz (personal computer).

5 Results

Distribution of Optimal Depth

In figure 1, we present histograms for the label distribution in our new dataset, i.e., how many nodes correspond to each optimal depth. Note that when training the depth classifier, we use under-sampling to counter the class imbalance. There is a visible change in distribution between the train and heldout node sets. One possible explanation to this difference is a distribution shift between the nodes, caused by the temporal data split used to create them (the train set consists of papers published earlier). This leads to unexpected performance in the data creation process over the heldout set.

Performance and Generalization of the Depth Classifier

For this experiment, let GNN-train, GNN-validation and GNN-test be the subgroups of nodes used to train the original network. We performed the data creation procedure (algorithm 2) to create three matching datasets, so that GNN-test is the heldout set of nodes used in the other experiments. For each dataset and for each max_depth in $\{3, 5, 7\}$, we compared different ML models via cross-validation. Table 1 shows the best model and its accuracy per max_depth. The accuracy is calculated over a test dataset.
While all models improved upon the random baseline, none improved by a significant margin; the objective classification accuracy is low. This contradicts our hypothesis, and will be discussed in section 6. The effect of the source dataset for data creation models the effect of distribution shift, and we can clearly see that the GNN-test generated dataset leads to worse performance compared to GNN-train. In addition, GNN-test always prefers the AdaBoost model while GNN-train leads to similar results for the different models. There is no clear relationship between max_depth and the best ML model.
Following the results in table 1, we considered adding the node's initial features to the centrality features before classification. The corresponding results are shown in table 2. The accuracy when training and evaluating on the GNN-test dataset improved marginally, but results on GNN-train decreased. The general trend hasn't changed.

Correlation heatmap

Figure 2 shows a correlation heatmap including the centrality measures and optimal depth labels. The bottom line (or right column) contains correlations of the features with the label, and we see little to no correlation. This result is in par with the poor performance of the depth classifiers we have seen in the previous experiment. Some possible explanations are discussed in the next section.

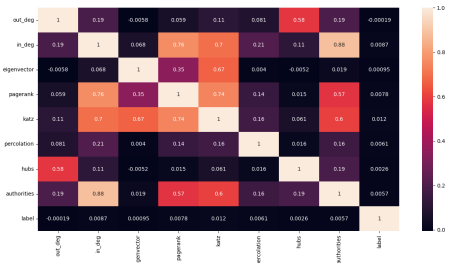


Figure 2: Correlation heatmap for centrality features and optimal depth

max_depth	GNN-train	GNN-validation	GNN-test	random model
3	B, 35.31	B, 36.6	B, 34.88	33.3
5	D, 23.04	C, 21.55	B, 21.58	20.0
7	C, 17.15	B, 18.29	B, 16.82	14.29

Table 1: best model and accuracy (%) over different sets and max_depths. Model index: (A) - decision tree, (B) - AdaBoost, (C) - Random Forest, (D) - MLP

max_depth	GNN-train	GNN-validation	GNN-test	random model
3	32.95	36.53	37.06	33.3
5	19.98	23.14	21.77	20.0
7	14.35	17.10	17.14	14.29

Table 2: best model and accuracy (%) over different sets and max_depths, trained and evaluated with centrality features as well as the graph features used for node classification. The model used is AdaBoost.

6 Discussion

In our experiments, specifically the depth classifier's performance and the correlation heatmap, we encountered surprising results; the ML models were not able to generalize well despite a reasonable amount of data (minimum of 30,000 data points in GNN-validation before train-test split), and the correlation between the centrality features to the label is non-existent. These results suggest the opposite from our hypothesis - that the centrality measures do not contain sufficient information to classify optimal depth. However, several other explanations are possible:

- Our GNN training procedure (algorithm 1) is not optimal. For example, the first message-passing layer is optimized by the loss from every classifier in every trining batch, while the last layer is only optimized by loss generated from the last classifier. We used SAGEconv layers with batch normalization, and possibly could have tweaked parameters better. In the extreme case of 7 layers, the minimal train loss is 1.0, and minimal test loss is 1.45. For comparison, a random model generates loss of 3.6.

- Temporal data split. Distribution shift may cause unpredictable labels in our created dataset.

- Limitation of centrality measures. Many informative centrality measures can't be efficiently computed over large graphs, such as betweenness and closeness. Extracting additional features may improve the results.

In retrospect, there are several arguments supporting the null hypothesis:

- Predicting the optimal depth may be even more difficult than solving the node classification problem. Thus, it will not be a simple function of centrality measures.

- The performance of the classifiers should intuitively depend on the embedding of nodes throughout the network's forward pass, which encapsulates their centrality features via message-passing. However, it may include important additional information, and therefore using only centrality is not enough to classify optimal depth.

7 Summary and future work

In this project, we have implemented a GNN training procedure which allowes node classification to be accomplished with a different number of message-passing steps per node. We used this idea to create a new dataset, in which the labels are the layer indices in which node classification loss is minimal. We attempted to find a connection between said labels and a set of 8 precalculated centrality measures of each node, by correlation and by training ML models for classification. We haven't found any reliable connection and deducted that additional information or stronger models are required to do so. Future work may investigate the following questions:

- Can we find a connection between optimal depth and the hidden node embeddings throughout the network's forward pass?
- Will similar results be found on other graph datasets?
- What is the effect of homophilic vs heterophilic input graph on the correctness of our hypothesis?
- What is the effect of the input graph's size on the correctness of our hypothesis?
- Can per-node-depth be succesfully applied to DL methods in other areas? such as Convolutional Neural Networks in Computer Vision tasks?

References

Do we need deep graph neural networks?

Uri Alon and Eran Yahav. 2020. On the bottleneck of graph neural networks and its practical implications. *arXiv preprint arXiv:2006.05205*.

Marco Gori, Gabriele Monfardini, and Franco Scarselli. 2005. A new model for learning in graph domains. In *Proceedings. 2005 IEEE international joint conference on neural networks*, volume 2, pages 729–734.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133.

Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. 2019. Deepgcns: Can gcns go as deep as cnns? In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9267–9276.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI conference on artificial intelligence*.

Kenta Oono and Taiji Suzuki. 2019. Graph neural networks exponentially lose expressive power for node classification. *arXiv preprint arXiv:1905.10947*.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. 2019. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint arXiv:1907.10903*.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80.

Hanqing Zeng, Muhan Zhang, Yinglong Xia, Ajitesh Srivastava, Andrey Malevich, Rajgopal Kannan, Viktor Prasanna, Long Jin, and Ren Chen. 2021. Decoupling the depth and scope of graph neural networks. *Advances in Neural Information Processing Systems*, 34:19665–19679.