PROGRESS REPORT: REPRESENTATION POWER OF GRAPH NEURAL NETWORK FOR LEARNING GRAPH PROPERTIES

Group G1: Zhenhan Huang, Anubhav Halder, Shiuli Subhra Ghosh, Alanta Kwok

ABSTRACT

Neural networks show great potential in various applications such as computer vision and natural language processing. At the same time, it becomes uncompetitive when trained on small-scale datasets such as tabular datasets that is still dominated by Gradient-Boosted Decision Trees (GBDT). There are few studies on the representation power of graph neural network in learning graph topology. This paper investigates and compares the effectiveness of different types of graph neural networks at learning and predicting different graph properties. Brief summary of results will be written here after all experiments conclude. As of currently, we have computed validation losses of properties under the GCN model.

1 INTRODUCTION

Graph neural networks (GNNs) have accumulated popularity in the last decade due to its versatility in applications including image classification, prediction of chemical/biological matters, cyber security analysis, etc. The application that our group has chosen to focus on this project is the representation power of neural networks on graph properties.

While neural networks are capable of analyzing graph data, the extent to which they can capture graph properties has not been extensively researched; recent work by Dehamy et al. (2019) has delved into the representation power of graph convolutional networks (GCNs) in graph topology, which documents the limitations and improvements of GCNs in learning graph moments.

For this project, we intend to extend on the paper's research by looking into how different architectures of GNNs perform at capturing various graph properties. Our goal is to demonstrate that the neural networks are able to compute NP-complete/hard graph properties to reasonable accuracy.

The dataset under investigation consists of synthetic graphs generated from the Erdos-Renyi (ER) and Barabási–Albert (BA) models. Analysis of graph properties are observed under traditional GNNs, GCNs and graph attention networks (GATs). The list of graph properties to be examined include clustering coefficients, path lengths between node pairs, page ranks, and more.

Through this work we will compare validation losses and their convergence for properties between different models and attempt to discover which models are best suited for computing specific properties.

2 GRAPH NEURAL NETWORK MODELS (GNN)

In machine learning, neural network models are used to learn and implement function τ that maps a graph G and one of its nodes n into a real vector in Euclidean space, i.e. $\tau : (G, v) \to \mathbb{R}^m$. (Gori et al., 2005) For traditional machine learning algorithms, the graph data is preprocessed to form a simpler reprsentation (e.g. vectorization.) Typically important information on graph topology is lost during preprocessing, thus may cause rseults to be wildly inaccurate. (Scarselli et al., 2009) Different models have been created to alleviate this problem by preserving the original graph structure of the data, including recursive neural networks (RNNs) and Markov chains.

Graph neural networks are an extension of RNNs that combines characteristics of both RNNs and random walk models that allow minimal preprocessing loss regardless of graph structure for graphfocused problems and can solve some node-focused problems without preprocessing. In the following subsections we will go over the different architectures of GNNs that are used to learn graph properties for this project.

2.1 Traditional GNNs

Similar to RNNs, traditional GNN models create node states and update them through iterative learning. Associated notation is defined here prior to further explanation: graphs are denoted as $G = (V, E)$ with corresponding sets of nodes V and edges E. Each node v has a label $\boldsymbol{h}_v \in \mathbb{R}^s$ which contains k features of the node, and all nodes directly connected to n through one edge (i.e. adjacent to v) are represented by the set $\delta(v)$. Denote the total number of nodes as $p = |V|$.

The explanation below for GNNs closely follows methods explained in Gori et al. (2005): The relationships between node v and its neighboring nodes and edges can be expressed as vector state $\boldsymbol{x}_v \in \mathbb{R}^d$, using the local transition function f_v which parametrically describes the dependence of v on its neighborhood:

$$\boldsymbol{x}_v = f_v(\boldsymbol{h}_v, \boldsymbol{x}_{\delta(v)}, \boldsymbol{h}_{\delta(v)}),\ v \in V \tag{1}$$

By concatenating all node states and labels into matrices $\boldsymbol{X}$ and $\boldsymbol{H}$ respectively, one can obtain an expression for the global transition function f:

$$\boldsymbol{X} = f(\boldsymbol{X}, \boldsymbol{H}) \tag{2}$$

The global transition function is generally represented by a multi-layer feed forward network where each layer l of output vectors $\boldsymbol{o}^l \in \mathbb{R}^{n_l}$ contains a weight matrix $\boldsymbol{W}^l \in \mathbb{R}^{n_l \times n_{l-1}}$, a bias vector $\boldsymbol{b}^l \in \mathbb{R}^{n_l}$, and a nonlinear activation function σ^l. Denoting the total number of layers as L, the layer-wise concatenated matrix of activation vectors $\boldsymbol{O}^l \in \mathbb{R}^{n_l \times p}$ for each layer l can be computed as

$$\boldsymbol{O}^l = \sigma^l(\boldsymbol{W}^l \boldsymbol{O}^{l-1} + \boldsymbol{b}^l), \quad l = 1, 2, \ldots, L \tag{3}$$

The global transition function is iteratively run to convergence for which $\boldsymbol{X}(t) \to \boldsymbol{X}$ with increasing values of iterate t. By the Banach fixed point theorem, the node states can be iteratively updated as

$$\boldsymbol{X}(t+1) = f(\boldsymbol{X}(t), \boldsymbol{H}) \tag{4}$$

where $\boldsymbol{O}^0 = \boldsymbol{X}(t)$ and $\boldsymbol{O}^L = \boldsymbol{X}(t+1)$ respectively.

The output vector each node $\boldsymbol{\theta}_v \in \mathbb{R}^m$ depends on the node's state and label, with a matrix representation of $\boldsymbol{\Theta} = g(\boldsymbol{X}, \boldsymbol{H})$ where $g : \mathbb{R}^{d \times p} \times \mathbb{R}^{s \times p} \to \mathbb{R}^{m \times p}$. In the context of iterations, for iterate t,

$$\boldsymbol{\Theta}(t) = g(\boldsymbol{X}(t), \boldsymbol{H}) \tag{5}$$

Given targets $\boldsymbol{y}_v \in \mathbb{R}^m$ for $v \in V$, the goal is to minimize the loss function $l(\boldsymbol{Y}, \boldsymbol{\Theta})$.

For this project, the initialization of $\mathbf{X}$ was chosen to be the node features (i.e. v-th column of $\mathbf{X}(0) = \mathbf{h}_v$ for all nodes v) for computational simplicity.

2.2 Graph convolution network models (GCN)

Graph convolution network models replace the feed forward neural network described in traditional GNNs with convolution neural networks. Under a first-order approximation of spectral filtering, one can obtain the layer-wise propogation that describes the outputs at layer l (Kipf & Welling, 2017):

$$\boldsymbol{O}^l = \sigma^l(\tilde{\boldsymbol{D}}^{-1/2} \tilde{\boldsymbol{A}} \tilde{\boldsymbol{D}}^{-1/2} \boldsymbol{W}^l \boldsymbol{O}^{l-1}) \tag{6}$$

where $\tilde{\boldsymbol{D}}^{-1/2}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-1/2}$ is a linear transformation of the normalized graph Laplacian. Here $\tilde{\boldsymbol{A}}$ is the sum of the adjacency matrix and the identity matrix, and the entries of the diagonal matrix $\tilde{\boldsymbol{D}}$ are the row-sums of the entries of $\tilde{\boldsymbol{A}}$, i.e. $(\tilde{\boldsymbol{D}})_{ii} = \sum_j \tilde{\boldsymbol{A}}_{ij}$. One may refer to Kipf & Welling (2017) on further details of spectral filtering.

2.3 Graph attention network models (GAT)

Graph attention network models are a modification of GNNs which include attention mechanisms in the neural network function f that attempts to improve output results by focusing on more important node features. Typically the model performs self-attention by automatically applying attention mechanisms at each layer.

Closely following the methodology of Velickovic et al. (2018), the self-attention mechanism at each layer is algebraically represented as a function that describes the dependence of one node from features of another node, denoted as $a_l : \mathbb{R}^{n_l} \times \mathbb{R}^{n_l} \to \mathbb{R}$ where the outputs are attention coefficients

$$e_{ij}^l = a_l(\boldsymbol{W}^l \boldsymbol{o}_i^l, \boldsymbol{W}^l \boldsymbol{o}_j^l) \quad \forall i \in V,\, j \in \delta(i) \tag{7}$$

The model used for this project represents the self-attention function as

$$a_l(\boldsymbol{W}^l \boldsymbol{o}_i^{l-1}, \boldsymbol{W}^l \boldsymbol{o}_j^{l-1}) = \text{LeakyReLU}\left((\boldsymbol{a}^l)^T [\boldsymbol{W}^l \boldsymbol{o}_i^{l-1} \| \boldsymbol{W}^l \boldsymbol{o}_j^{l-1}]\right) \tag{8}$$

where $\boldsymbol{a}^l \in \mathbb{R}^{2n_l}$ is the self-attention weight vector, $[\boldsymbol{W}^l \boldsymbol{o}_i^{l-1} \| \boldsymbol{W}^l \boldsymbol{o}_j^{l-1}]$ is the concatenation of the vectors representing linear transformation of output features from the previous layer of nodes i and j, and the LeakyReLU function is expressed as

$$\text{LeakyReLU}(x) = \begin{cases} x & x \geq 0 \\ 0.01x & x < 0 \end{cases} \tag{9}$$

The attention coefficients are then normalized through the use of the softmax function:

$$\alpha_{ij}^l = \text{softmax}(e_{ij}^l) = \frac{e_{ij}^l}{\sum_{k \in \delta(i)} e_{ik}^l} \tag{10}$$

For stability purposes, GATs generally take a multihead approach where multiple independent attention mechanisms are performed simultaneously in one layer. For K independent mechanisms, the layer-wise output vectors are computed as

$$\boldsymbol{o}_i^l = \begin{cases} \|_{k=1}^K \sigma\left(\sum_{j\in\delta(i)} \alpha_{ij}^{(k,l)} \mathbf{W}^{(k,l)} \boldsymbol{o}_i^{l-1}\right) & l \neq L \\ \sigma\left(\frac{1}{K}\sum_{k=1}^K \sum_{j\in\delta(i)} \alpha_{ij}^{(k,l)} \mathbf{W}^{(k,l)} \boldsymbol{o}_i^{l-1}\right) & l = L \end{cases} \tag{11}$$

where $\alpha_{ij}^{(k,l)}$ represents the normalized attention coefficient between neighboring nodes i and j for independent mechanism k on layer l, and $\|_{k=1}^K$ refers to the concatenation of vectors across all independent mechanisms (thus $\boldsymbol{o}_i^l \in \mathbb{R}^{kn_l}$ for $l \neq L$, as opposed to $\boldsymbol{o}_i^L \in \mathbb{R}^{n_L}$.)

3 Experimental methods

2500 undirected training graphs of 64 nodes were generated each from Erdos-Renyi and Barabási–Albert models in the form of adjacency matrices using graph generation functions from the NetworkX package. Of the total of 5000 graphs, 4000 of them were randomly chosen to be the training set, with the remaining 1000 allocated to the test set.

The training graphs were inputted into the neural network models to learn specific key graph properties which provide a strong description of the graph structure. All neural network models were trained for 2000 epochs. After the models were trained, the test graphs were inputted and validation losses were computed for both unnormalized and normalized property values for 2000 epochs.

Property values were computed through processing the output graphs from neural networks over property functions through the NetworkX packages. The normalized set of property values of each node were computed relative to the mean and standard deviation of the unnormalized properties for all nodes in each graph. The validation loss was computed as the average of mean-squared error loss between true graph property values and predicted graph property values for all graphs. For each graph, the mean-squared error loss can be expressed as

$$l_{\text{MSE}}(\boldsymbol{Y}, \boldsymbol{\Theta}) = \frac{1}{p} \sum_{v=1}^{p} \|\boldsymbol{y}_v - \boldsymbol{\theta}_v\|_2^2 \tag{12}$$

It should be noted that no regularization has been used for computing the loss.

3.1 Graph generation models

3.1.1 Erdos-Renyi (ER) model

The Erdos-Renyi model, commonly referred to as the $G(n, p)$ model, generates a random graph with n vertices where each edge between two vertices has probability $p \in [0, 1]$ to exist. The graphs generated from the model are also commonly known as binomial graphs since the degree of each node v follows a binomial distribution (Erdos & Renyi, 1960):

$$P(\deg(v) = C) = \binom{n-1}{C} p^C (1-p)^{n-1-C} \tag{13}$$

In the context of this project, the probability variable p is randomly generated for each graph.

3.1.2 Barabási–Albert (BA) model

The Barabási–Albert model generates scale-free network graphs based on an algorithm that includes a preferential attachment mechanism. Scale-free network graphs are generalized by node degrees having a probability distribution that approximately follows a power law, i.e. $P(\deg(v) = C) \sim C^{-\gamma}$ for positive constant $\gamma \approx 3$ (Barabási & Albert, 1999). Preferential attachment is the tendency of new nodes to connect to existing nodes with a larger number of connections, and is a key property of real networks (Barabási & Albert, 2002).

The generation algorithm for a graph of n nodes can be summarized as follows (Barabási & Albert, 1999):

1. Initialize with a connected graph G of m_0 nodes with $m_0 < n$.
2. Generate a new node in G that is connected to m existing nodes for some constant $m \leq m_0$. To demonstrate preferential attachment, the probability that the new node is connected to existing node v_i is given by the equation

$$p_i = \frac{\deg(v_i)}{\sum_{v_j \in N(G)} \deg(v_j)} \tag{14}$$

3. Repeat Step 2 until the graph has n nodes.

In the context of this project, the integer variable $m \in [0, n]$ is randomly generated, with the initial graph being a randomly generated star graph with $m + 1$ nodes.

3.2 Graph properties

As of currently, graph properties predicted by the models include:

- **Degree centrality** (hereby referred to as centrality): for any node v, this refers to the ratio of the number of nodes in $\delta(v)$ to the number of nodes in set $V\backslash v$.
- **Clustering coefficient**: For graphs with unweighted edges, the clustering coefficient of any node v refers to the proportion of possible triangles that pass through v which exist within the graph.
- **Eccentricity**: the maximum of path lengths between v and any other node in the graph.
- **Page rank**: The page rank of nodes are determined by applying the PageRank search engine algorithm to order the importance of nodes using power iterations. In the context of this project, through the NetworkX package, the page ranks are computed using an eigenvector algorithm based on convergence of power iterations; see (Perra & Fortunato, 2008) for algorithm details.
- **Average neighbor degree**: the mean of the degrees of all nodes in $\delta(v)$.
- **Average shortest path length**: the mean of the shortest path lengths between node v and all other nodes in the graph.

As of April 7, 2023, all the properties listed focus on local properties of the node and its neighbors.

4 RESULTS (CURRENT PROGRESS)

4.1 GCN RESULTS

Properties were computed under 1-layer, 2-layer and 4-layer GCNs under the logistic sigmoid activation function. Figure 1 shows loss comparisons between GCN models of different layers for learning different graph properties.

For unnormalized properties, the 4-layer model performs significantly better at predicting eccentricity compared to the 1-layer and 2-layer models, requiring much less epochs to reach a loss of $\leq 10^{-2}$. It also outperforms the other models at predicting clustering coefficients. The convergence rates of the 2-layer and 4-layer models at for losses other properties are similar, whereas the 1-layer model is unable to achieve the same level of validation loss compared to the 2-layer and 4-layer models for the majority of properties.

For normalized properties, all models exhibit similar convergence rates for normalized centrality and page rank validation losses. Validation losses of the 2-layer and 4-layer models approached similar values for the majority of normalized properties at large number of epochs with the exception of the neighbor degree for which the 4-layer model has a significantly smaller validation loss compared to the other two models.

From the comparison of the performance of the architectures, the 4-layer model generally has the smallest validation losses for both unnormalized and normalized properties, which is to be expected since the neural network undergoes multiple layers of learning for better accuracy. It is worth noting that the 2-layer model has similar accuracy in measuring normalized properties, thus is a reasonable choice for time-efficient computations.

The effects of normalization can be further visualized by comparing the losses for the 4-layer model for each property, as demonstrated in Figure 2. Validation losses of normalized centrality, shortest path length and page rank values converge much faster than that of their unnormalized variants. Moreover, validation losses for unnormalized properties appear to fluctuate at large number of epochs, whereas this characteristic is absent for their normalized variants.

5 FUTURE WORK

As of April 7, 2023, we are currently obtaining results for traditional GNN and GAT models. Once results are computed, losses of properties for each model will be compared against each other to discover and possibly explain which models are best suited for learning specific graph properties.

Furthermore, we are considering analysis of more computationally complex graph properties that provide a more global description of graph structure, e.g. maximum induced path, k-degree spanning

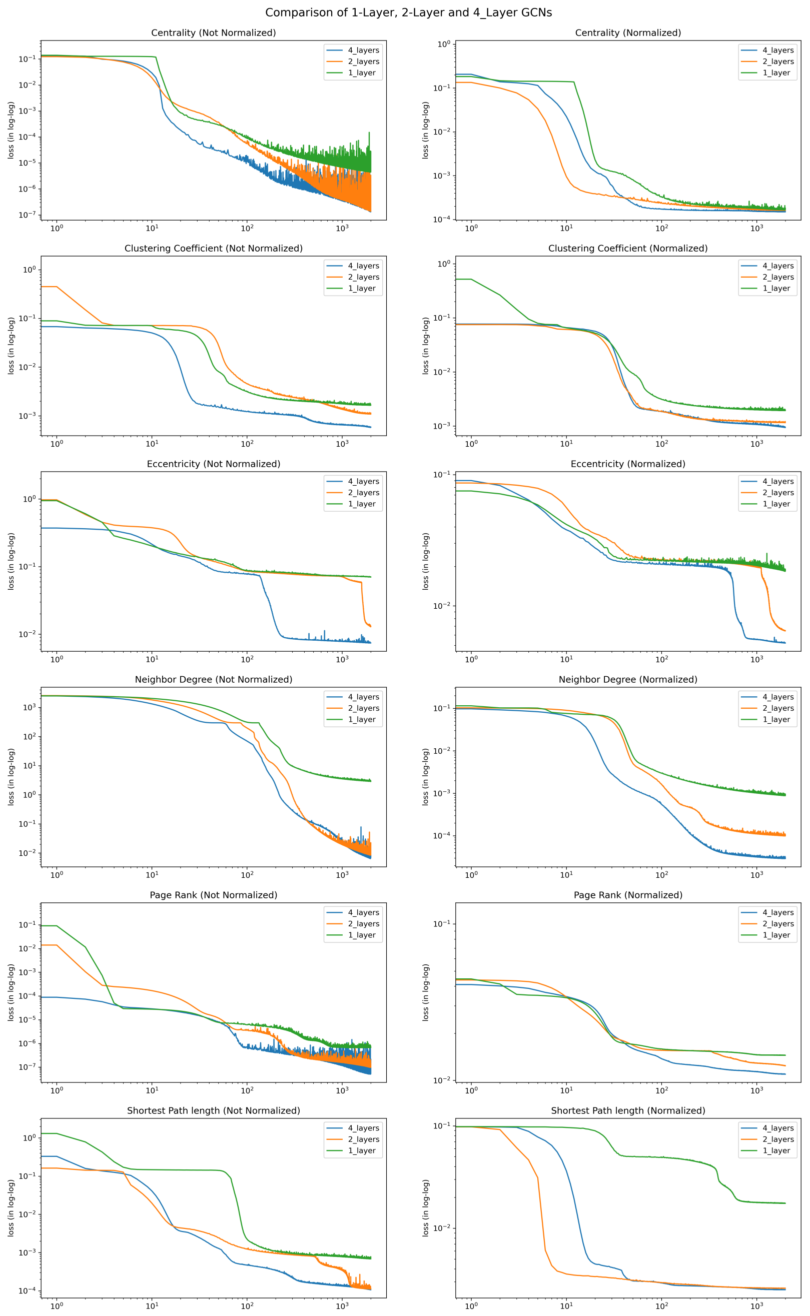

Figure 1: Loss comparisons between different GCN architectures for both unnormalized and normalized graph properties.

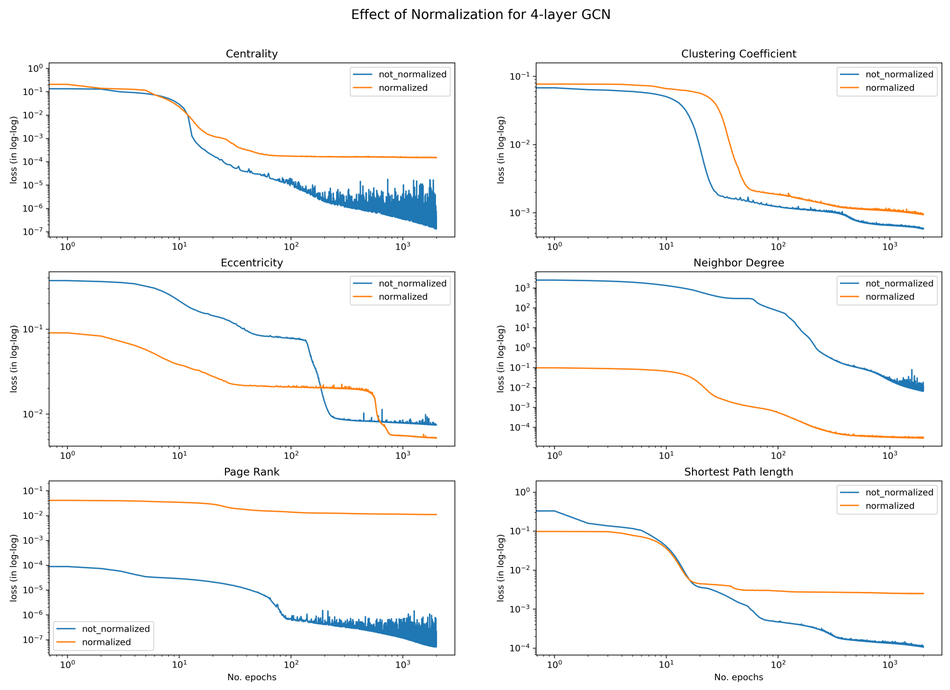

Figure 2: Comparisons of loss for GCN 4-layer model for both unnormalized and normalized graph properties.

trees, existence of Hamiltonian paths/cycles, etc. to further achieve our goal of demonstrating neural networks have the capability to compute NP-complete/hard graph properties.

6 Conclusions

This section is to be written after all experiments conclude.

References

Albert-László Barabási and Reka Albert. Emergence of scaling in random networks. *Science*, 286, 1999.

Albert-László Barabási and Reka Albert. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74, 2002.

Nima Dehamy, Albert-László Barabási, and Rose Yu. Understanding the representation power of graph neural networks in learning graph topology. 2019. URL `https://arxiv.org/pdf/1907.05008.pdf`.

Paul Erdos and Alfred Renyi. On the evolution of random graphs. 1960. URL `https://static.renyi.hu/~p_erdos/1960-10.pdf`.

Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. 2005.

Thomas Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. 2017. URL `https://arxiv.org/pdf/1609.02907.pdf%EF%BC%89`.

Nicola Perra and Santo Fortunato. Spectral centrality measures of complex networks. 2008. URL `https://arxiv.org/pdf/0805.3322.pdf`.

Franco Scarselli, Maro Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. 2009. URL `https://ro.uow.edu.au/cgi/viewcontent.cgi?article=10501&context=infopapers`.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. 2018. URL `https://arxiv.org/pdf/1710.10903.pdf`.