Understanding the Representation Power of Graph Neural Networks in Learning Graph Topology

Nima Dehmamy[*]
CSSI, Kellogg School of Management
Northwestern University, Evanston, IL
nimadt@bu.edu

Albert-László Barabási[†]
Center for Complex Network Research,
Northeastern University, Boston MA
alb@neu.edu

Rose Yu
Khoury College of Computer Sciences,
Northeastern University, Boston, MA
roseyu@northeastern.edu

Abstract

To deepen our understanding of graph neural networks, we investigate the representation power of Graph Convolutional Networks (GCN) through the looking glass of *graph moments*, a key property of graph topology encoding path of various lengths. We find that GCNs are rather restrictive in learning graph moments. Without careful design, GCNs can fail miserably even with multiple layers and nonlinear activation functions. We analyze theoretically the expressiveness of GCNs, concluding that a modular GCN design, using different propagation rules with residual connections could significantly improve the performance of GCN. We demonstrate that such modular designs are capable of distinguishing graphs from different graph generation models for surprisingly small graphs, a notoriously difficult problem in network science. Our investigation suggests that, depth is much more influential than width, with deeper GCNs being more capable of learning higher order graph moments. Additionally, combining GCN modules with different propagation rules is critical to the representation power of GCNs.

1 Introduction

The surprising effectiveness of graph neural networks [17] has led to an explosion of interests in graph representation learning, leading to applications from particle physics [12], to molecular biology [37] to robotics [4]. We refer readers to several recent surveys [7, 38, 33, 14] and the references therein for a non-exhaustive list of the research. Graph convolution networks (GCNs) are among the most popular graph neural network models. In contrast to existing deep learning architectures, GCNs are known to contain fewer number of parameters, can handle irregular grids with non-Euclidean geometry, and introduce relational inductive bias into data-driven systems. It is therefore commonly believed that graph neural networks can learn arbitrary representations of graph data.

Despite their practical success, most GCNs are deployed as black boxes feature extractors for graph data. It is not yet clear to what extent can these models capture different graph features. One prominent feature of graph data is *node permutation invariance*: many graph structures stay the same

[*]work done when at Center for Complex Network Research, Northeastern University, Boston, MA

[†]Center for Cancer Systems Biology, Dana Farber Cancer Institute, Boston MA, Brigham and Women's Hospital, Harvard Medical School, Boston MA, Center for Network Science, Central European University, Budapest, Hungary

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

under relabelling or permutations of the nodes. For instance, people in a friendship network may be following a similar pattern for making friends, in similar cultures. To satisfy permutation invariance, GCNs assign global parameters to all the nodes by which significantly simplifies learning. But such efficiency comes at the cost of expressiveness: GCNs are *not* universal function approximators [34]. We use GCN in a broader sense than in [20], allowing different propagation rules (see below (4)).

To obtain deeper understanding of graph neural networks, a few recent work have investigated the behavior of GCNs including expressiveness and generalizations. For example, [28] showed that message passing GCNs can approximate measurable functions in probability. [34, 24, 25] defined expressiveness as the capability of learning multi-set functions and proved that GCNs are at most as powerful as the Weisfeiler-Lehman test for graph isomorphism, but assuming GCNs with infinite number of hidden units and layers. [32] analyzed the generalization and stability of GCNs, which suggests that the generalization gap of GCNs depends on the eigenvalues of the graph filters. However, their analysis is limited to a single layer GCN for semi-supervised learning tasks. Up until now, the representation power of multi-layer GCNs for learning graph topology remains elusive.

In this work, we analyze the representation power of GCNs in learning graph topology using *graph moments*, capturing key features of the underlying random process from which a graph is produced. We argue that enforcing node permutation invariance is restricting the representation power of GCNs. We discover pathological cases for learning graph moments with GCNs. We derive the representation power in terms of number of hidden units (width), number of layers (depths), and propagation rules. We show how a modular design for GCNs with different propagation rules significantly improves the representation power of GCN-based architectures. We apply our modular GCNs to distinguish different graph topology from small graphs. Our experiments show that depth is much more influential than width in learning graph moments and combining different GCN modules can greatly improve the representation power of GCNs. [3]

In summary, our contributions in this work include:

- We reveal the limitations of graph convolutional networks in learning graph topology. For learning graph moments, certain designs GCN completely fails, even with multiple layers and non-linear activation functions.
- we provide theoretical guarantees for the representation power of GCN for learning graph moments, which suggests a strict dependence on the depth whereas the width plays a weaker role in many cases.
- We take a modular approach in designing GCNs that can learn a large class of node permutation invariant function of of the graph, including non-smooth functions. We find that having different graph propagation rules with residual connections can dramatically increase the representation power of GCNs.
- We apply our approach to build a "graph stethoscope": given a graph, classify its generating process or topology. We provide experimental evidence to validate our theoretical analysis and the benefits of a modular approach.

Notation and Definitions A graph is a set of N nodes connected via a set of edges. The adjacency matrix of a graph A encodes graph topology, where each element A_{ij} represents an edge from node i to node j. We use AB and $A \cdot B$ (if more than two indices may be present) to denote the matrix product of matrices A and B. All multiplications and exponentiations are matrix products, unless explicitly stated. Lower indices A_{ij} denote i, jth elements of A, and A_i means the ith row. A^p denotes the pth matrix power of A. We use $a^{(m)}$ to denote a parameter of the mth layer.

2 Learning Graph Moments

Given a collection of graphs, produced by an unknown random graph generation process, learning from graphs requires us to accurately infer the characteristics of the underlying generation process. Similar to how moments $\mathbb{E}[X^p]$ of a random variable X characterize its probability distribution, graph moments [5, 23] characterize the random process from which the graph is generated.

[3] All code and hyperparameters are available at `https://github.com/nimadehmamy/Understanding-GCN`

2.1 Graph moments

In general, a pth order graph moment M_p is the ensemble average of an order p polynomial of A

$$M_p(A) = \prod_{q=1}^{p} (A \cdot W_q + B_q) \tag{1}$$

with W_q and B_q being $N \times N$ matrices. Under the constraint of node permutation invariance, W_q must be either proportional to the identity matrix, or a uniform aggregation matrix. Formally,

$$M(A) = A \cdot W + B, \quad \text{Node Permutation Invariance} \Rightarrow \quad W, B = cI, \quad \text{or} \quad W, B = c\mathbf{1}\mathbf{1}^T \tag{2}$$

where $\mathbf{1}$ is a vector of ones. Graph moments encode topological information of a graph and are useful for graph coloring and Hamiltonicity. For instance, graph power A^p_{ij} counts the number of paths from node i to j of length p. For a graph of size N, A has N eigenvalues. Applying eigenvalue decomposition to graph moments, we have $\mathbb{E}[A^p] = \mathbb{E}[(V^T \Lambda U)^p]) = V^T \mathbb{E}[\Lambda^p] U$. Graphs moments correspond to the distribution of the eigenvalues Λ, which are random variables that characterize the graph generation process. Graph moments are node permutation invariant, meaning that relabelling of the nodes will not change the distribution of degrees, the paths of a given length, or the number of triangles, to name a few. The problem of learning graph moments is to learn a functional approximator F such that $F : A \to M_p(A)$, while preserving node permutation invariance.

Different graph generation processes can depend on different orders of graph moments. For example, in Barabási-Albert (BA) model [1], the probability of adding a new edge is proportional to the degree, which is a first order graph moment. In diffusion processes, however, the stationary distribution depends on the normalized adjacency matrix $\hat{A}$ as well as its symmetrized version $\hat{A}_s$, defined as follows:

$$D_{ij} \equiv \delta_{ij} \sum_k A_{ik} \qquad \hat{A} \equiv D^{-1} A \qquad \hat{A}_s \equiv D^{-1/2} A D^{-1/2} \tag{3}$$

which are *not* smooth functions of A and have no Taylor expansion in A, because of the inverse D^{-1}. Processes involving D^{-1} and A are common and per (2) D and $\mathrm{Tr}[A]$ are the only node permutation invariant first order moments of A. Thus, in order to approximate more general node permutation invariant $F(A)$, it is crucial for a graph neural network to be able to learn moments of A, $\hat{A}$ and $\hat{A}_s$ simultaneously. In general, non-smooth functions of A can depend on A^{-1}, which may be important for inverting a diffusion process. We will only focus on using A, $\hat{A}$ and $\hat{A}_s$ here, but all argument hold also if we include A^{-1}, $\hat{A}^{-1}$ and $\hat{A}_s^{-1}$ as well.

2.2 Learning with Fully Connected Networks

Consider a toy example of learning the first order moment. Given a collection of graphs with $N = 20$ nodes, the inputs are their adjacency matrices A, and the outputs are the node degrees $D_i = \sum_{j=1}^{N} A_{ij}$. For a fully connected (FC) neural network, it is a rather simple task given its universal approximation power [19]. However, a FC network treats the adjacency matrices as vector inputs and ignores the underlying graph structures, it needs a large amount of training samples and many parameters to learn properly.

Fig. 1 shows the mean squared error (MSE) of a single layer FC network in learning the first order moments. Each curve corresponds to different number of training samples, ranging from 500–10,000. The horizontal axis shows the number of hidden units. We can see that even though the network can learn the moments properly reaching an MSE of $\approx 10^{-4}$, it requires the same order of magnitude of hidden units as the number of nodes in the graph, and at least $1,000$ samples. Therefore, FC networks are quite inefficient for learning graph moments, which motivates us to look into more power alternatives: graph convolution networks.

2.3 Learning with Graph Convolutional Networks

We consider the following class of graph convolutional networks. A single layer GCN propagates the node attributes h using a function $f(A)$ of the adjacency matrix and has an output given by

$$F(A, h) = \sigma\left(f(A) \cdot h \cdot W + b\right) \tag{4}$$

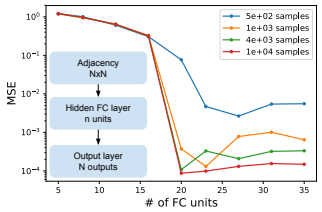

Figure 1: Learning graph moments (Erdős-Rényi graph) with a single fully-connected layer. Best validation MSE w.r.t number of hidden units n and the number of samples in the training data (curves of different colors).

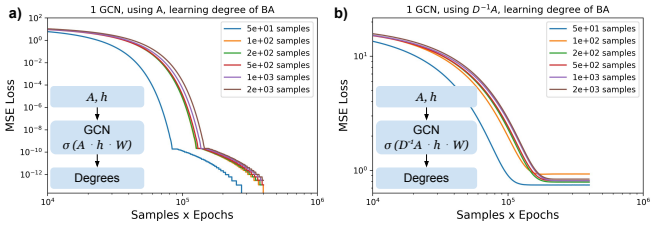

Figure 2: Learning the degree of nodes in a graph with a single layer of GCN. When the GCN layer is designed as $\sigma(A \cdot h \cdot W)$ with linear activation function $\sigma(x) = x$, the network easily learns the degree (a). However, if the network uses the propagation rule as $\sigma(D^{-1}A \cdot h \cdot W)$, it fails to learn degree, with very high MSE loss (b). The training data were instances of Barabasi-Albert graphs (preferential attachment) with $N = 20$ nodes and $m = 2$ initial edges.

where f is called the propagation rule, h_i is the attribute of node i, W is the weight matrix and b is the bias. As we are interested in the graph topology, we ignore the node attributes and set $h_i = 1$. Note that the weights W are only coupled to the node attributes h but not to the propagation rule $f(A)$. The definition in Eqn (4) covers a broad class of GCNs. For example, GCN in [20] uses $f = D^{-1/2}AD^{-1/2}$. GraphSAGE [16] mean aggregator is equivalent to $f = D^{-1}A$. These architectures are also special cases of Message-Passing Neural Networks [13].

We apply a single layer GCN with different propagation rules to learn the node degrees of BA graphs. With linear activation $\sigma(x) = x$, the solution for learning node degrees is $f(A) = A$, $W = 1$ and $b = 0$. For high-order graph moments of the form $M_p = \sum_j (A^p)_{ij}$, a single layer GCN has to learn the function $f(A) = A^p$. As shown in Figure 2, a single layer GCN with $f(A) = A$ can learn the degrees perfectly even with as few as 50 training samples for a graph of $N = 20$ nodes (Fig. 2a). Note that GCN only requires 1 hidden unit to learn, which is much more efficient than the FC networks. However, if we set the learning target as $f(A) = D^{-1}A$, the same GCN completely fails at learning the graph moments regardless of the sample size, as shown in Fig. 2b. This demonstrates the limitation of GCNs due to the permutation invariance constraint. Next we analyze this phenomena and provide theoretical guarantees for the representation power of GCNs.

3 Theoretical Analysis

To learn graph topology, fully connected layers require a large number of hidden units. The following theorem characterizes the representation power of fully connected neural network for learning graph moments in terms of number of nodes N, order of moments p and number of hidden units n.

Theorem 1. *A fully connected neural network with one hidden layer requires $n > O(C_f^2) \sim O(p^2 N^{2q})$ number of neurons in the best case with $1 \leq q \leq 2$ to learn a graph moment of order p for graphs with N nodes. Additionally, it also needs $S > O(nd) \sim O\left(p^2 N^{2q+2}\right)$ number of samples to make the learning tractable.*

Clearly, if a FC network fully parameterizes every element in a $N \times N$ adjacency matrix A, the dimensions of the input would have to be $d = N^2$. If the FC network allows weight sharing among nodes, the input dimension would be $d = N$. The Fourier transform of a polynomial function of order p with $O(1)$ coefficients will have an L_1 norms of $C_f \sim O(p)$. Using Barron's result [2] with $d = N^q$, where $1 \leq q \leq 2$ and set the $C_f \sim O(p)$, we can obtain the approximation bound.

In contrast to fully connected neural networks, graph convolutional networks are more efficient in learning graph moments. A graph convolution network layer without bias is of the form:

$$F(A, h) = \sigma(f(A) \cdot h \cdot W) \tag{5}$$

Permutation invariance restricts the weight matrix W to be either proportional to the identity matrix, or a uniform aggregation matrix, see Eqn. (2). When $W = cI$, the resulting graph moment $M_p(A)$ has exactly the form of the output of a p layer GCN with linear activation function.

We first show, via an explicit example, that a $n < p$ layer GCN by stacking layers of the form in Eqn. (5) cannot learn pth order graph moments.

Lemma 1. *A graph convolutional network with $n < p$ layers cannot, in general, learn a graph moment of order p for a set of random graphs.*

We prove this by showing a counterexample. Consider a directed graph of two nodes with adjacency matrix $A = \begin{pmatrix} 0 & a \\ b & 0 \end{pmatrix}$. Suppose we want to use a single layer GCN to learn the second order moment $f(A)_i = \sum_j (A^2)_{ij} = \sum_k A_{ik} D_k$. The node attributes h_{il} are decoupled from the propagation rule $f(A)_i$. Their values are set to ones $h_{il} = 1$, or any values independent of A. The network tries to learn the weight matrix $W_{l\mu}$ and has an output $h^{(1)}$ of the form

$$h^{(1)}_{i\mu} = \sigma\left(A \cdot h \cdot W\right)_{i\mu} = \sigma\left(\sum_{j,l} A_{ij} h_{jl} W_{l\mu}\right), \tag{6}$$

For brevity, define $V_{i\mu} \equiv \sum_l h_{il} W_{l\mu}$. Setting the output $h^{(1)}$ to the desired function $A \cdot D$, with components $h^{(1)}_{1\mu} = h^{(1)}_{2\mu} = ab$, (hence μ can only be 1) and plugging in A, the two components of the output become

$$h^{(1)}_{1\mu} = \sigma\left(D_1 V_{1\mu}\right) = \sigma\left(a V_{1\mu}\right) = ab \qquad h^{(1)}_{2\mu} = \sigma\left(D_2 V_{2\mu}\right) = \sigma\left(b V_{2\mu}\right) = ab. \tag{7}$$

which must be satisfied $\forall a, b$. But it's impossible to satisfy $\sigma\left(a V_{1\mu}\right) = ab$ for $(a, b) \in \mathbb{R}^2$ with $V_{1\mu}$ and $\sigma(\cdot)$ independent of a, b. □

Proposition 1. *A graph convolutional network with n layers, and no bias terms, in general, can learn $f(A)_i = \sum_j \left(A^p\right)_{ij}$ only if $n = p$ or $n > p$ if the bias is allowed.*

If we use a two layer GCN to learn a first order moment $f(A)_i = \sum_j A_{ij} = D_i$, for the output of the second layer $h^{(2)}_{i\nu}$ we have

$$h^{(2)} = \sigma^{(2)}\left(A \cdot \sigma^{(1)}\left(A \cdot h \cdot W^{(1)}\right) \cdot W^{(2)}\right),\ h^{(2)}_{1\nu} = \sigma^{(2)}\left(a \sum_\mu \sigma^{(1)}\left(b V^{(1)}_{2\mu}\right) W^{(2)}_{\mu\nu}\right) = a \tag{8}$$

Again, since this must hold for any value of a, b and ν, we see that $h^{(2)}_{1\nu}$ is a function of b through the output of the first layer $h^{(1)}_{2\mu}$. Thus $h^{(2)}_{1\nu} = a$ can only be satisfied if the first layer output is a constant. In other words, only if the first layer can be bypassed (e.g. if the bias is large and weights are zero) can a two-layer GCN learn the first order moment. □

This result also generalizes to multiple layers and higher order moments in a straightforward fashion. For GCN with linear activation, a similar argument shows that when the node attributes h are not implicitly a function of A, in order to learn the function $\sum_j \left(A^p\right)_{ij}$, we need to have exactly $n = p$ GCN layers, without bias. With bias, a feed-forward GCN with $n > p$ layers can learn single term order p moments such as $\sum_j \left(A^p\right)_{ij}$. However, since it needs to set the some weights of $n - p$ layers to zero, it can fail in learning mixed order moments such as $\sum_j (A^q + A^p)_{ij}$.

To allow GCNs with very few parameters to learn mixed order moments, we introduce residual connections [18] by concatenating the output of every layer $[h^{(1)}, \ldots, h^{(m)}]$ to the final output of the network. This way, by applying an aggregation layer or a FC layer which acts the same way on the output for every node, we can approximate any polynomial function of graph moments. Specifically, the final $N \times d^o$ output $h^{(final)}$ of the aggregation layer has the form

$$h^{(final)}_{i\mu} = \sigma\left(\sum_{m=1}^{n} a^{(m)}_\mu \cdot h^{(m)}_i\right), \qquad h^{(m)} = \sigma(A \cdot h^{(m-1)} \cdot W^{(m)} + b^{(m)}), \tag{9}$$

where $\cdot$ acts on the output channels of each output layers. The above results lead to the following theorem which guarantees the representation power of multi-layer GCNs with respect to learning graph moments.

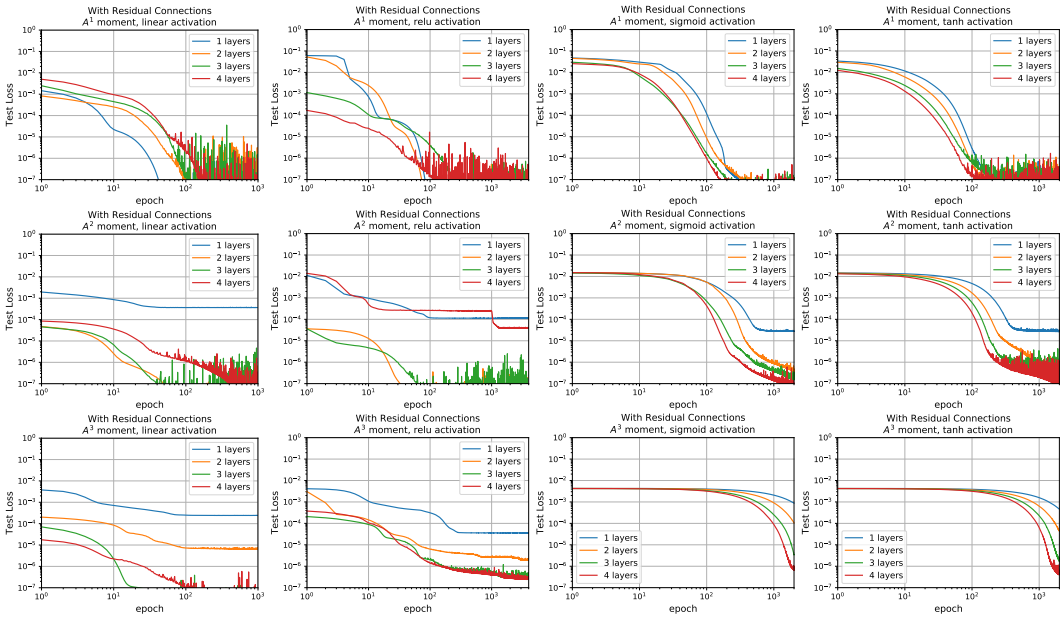

Figure 4: Test loss over number of epochs for learning first (top), second (middle) and third (bottom) order graph moments $M_p(A) = \sum_j (A^p)_{ij}$, with varying number of layers and different activation functions. A multi-layer GCN with residual connections is capable of learning the graph moments when the number of layers is at least the target order of the graph moments. The graphs are from our synthetic graph dataset described in Sec. 6.

Theorem 2. *With the number of layers n greater or equal to the order p of a graph moment $M_p(A)$, graph convolutional networks with residual connections can learn a graph moment M_p with $O(p)$ number of neurons, independent of the size of the graph.*

Theorem 2 suggests that the representation power of GCN has a strong dependence on the number of layers (depth) rather than the size of the graph (width). It also highlights the importance of residual connections. By introducing residual connections into multiple GCN layers, we can learn any polynomial function of graph moments with linear activation. Interestingly, Graph Isomophism Network (GIN) proposed in [34] uses the following propagation rule:

$$F(A, h) = \sigma\left([(1+\epsilon)I + A] \cdot h \cdot W\right) \tag{10}$$

which is a special case of our GCN with one residual connection between two modules.

4 Modular GCN Design

In order to overcome the limitation of the GCNs in learning graph moments, we take a modular approach to GCN design. We treat different GCN propagation rules as different "modules" and consider three important GCN modules (1) $f_1 = A$ [22] (2) $f_2 = D^{-1}A$ [20], and (3) $f_3 = D^{-1/2}AD^{-1/2}$ [16]. Figure 3a) shows the design of a single GCN layer where we combine three different GCN modules. The output of the modules are concatenated and fed into a node-wise FC layer. Note that our design is different from the multi-head attention mechanism in Graph Attention Network [31] which uses the same propagation rule for all the modules.

However, simply stacking GCN layers on top of each other in a feed-forward fashion is quite restrictive, as shown by our theoretical analysis for multi-layer GCNs. Different

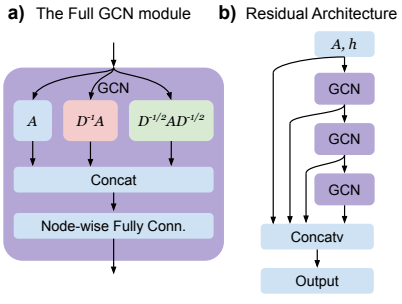

Figure 3: GCN layer (a), using three different propagation rules and a node-wise FC layer. Using residual connections (b) allows a n-layer modular GCN to learn any polynomial function of order n of its constituent operators.

propagation rules cannot be written as Taylor expansions of each other, while all of them are important in modeling the graph generation process. Hence, no matter how many layers or how non-linear the activation function gets, multi-layer GCN stacked in a feed-forward way cannot learn network moments whose order is not precisely the number of layers. If we add residual connections from the output of every layer to the final aggregation layer, we would be able to approximate any polynomial functions of graph moments. Figure 3b) shows the design of a muli-layer GCN with residual connections. We stack the modular GCN layer on top of each other and concatenate the residual connections from every layer. The final layer aggregates the output from all previous layers, including residual connections.

We measure the representation power of GCN design in learning different orders of graph moments $M_p(A) = \sum_j (A^p)_{ij}$ with $p = 1, 2, 3$. Figure 4 shows the test loss over number of epochs for learning first (top), second (middle) and third (bottom) order graph moments. We vary the number of layers from 1 to 4 and test with different activation functions including linear, ReLU, sigmoid and tanh. Consistent with the theoretical analysis, we observe that whenever the number of layers is at least the target order of the graph moments, a multi-layer GCN with residual connections is capable of learning the graph moments. Interestingly, Jumping Knowledge (JK) Networks [35] showed similar effects of adding residual connections for Message Passing Graph Neural Networks.

Our modular approach demonstrates the importance of architectural design when using specialized neural networks. Due to permutation invariance, feed-forward GCNs are quite limited in their representation power and can fail at learning graph topology. However, with careful design including different propagation rules and residual connections, it is possible to improve the representation power of GCNs in order to capture higher order graph moments while preserving permutation invariance.

5 Related Work

Graph Representation Learning There has been increasing interest in deep learning on graphs, see e.g. many recent surveys of the field [7, 38, 33]. Graph neural networks [22, 20, 17] can learn complex representations of graph data. For example, Hopfield networks [28, 22] propagate the hidden states to a fixed point and use the steady state representation as the embedding for a graph; Graph convolution networks [8, 20] generalize the convolutional operation from convolutional neural networks to learn from geometric objects beyond regular grids. [21] proposes a deep architecture for long-term forecasting of spatiotemporal graphs. [37] learns the representations for generating random graphs sequentially using an adversarial loss at each step. Despite practical success, deep understanding and theoretical analysis of graph neural networks is still largely lacking.

Expressiveness of Neural Networks Early results on the expressiveness of neural networks take a highly theoretical approach, from using functional analysis to show universal approximation results [19], to studying network VC dimension [3]. While these results provided theoretically general conclusions, they mostly focus on single layer shallow networks. For deep fully connected networks, several recent papers have focused on understanding the benefits of depth for neural networks [11, 29, 28, 27]) with specific choice of weights. For graph neural networks, [34, 24, 25] prove the equivalence of a graph neural network with Weisfeiler-Lehman graph isomorphism test with infinite number of hidden layers. [32] analyzes the generalization and stability of GCNs, which depends on eigenvalues of the graph filters. However, their analysis is limited to a single layer GCN in the semi-supervised learning setting. Most recently, [10] demonstrates the equivalence between infinitely wide multi-layer GNNs and Graph Neural Tangent Kernels, which enjoy polynomial sample complexity guarantees.

Distinguishing Graph Generation Models Understanding random graph generation processes has been a long lasting interest of network analysis. Characterizing the similarities and differences of generation models has applications in, for example, graph classification: categorizing a collections of graphs based on either node attributes or graph topology. Traditional graph classification approaches rely heavily on feature engineering and hand designed similarity measures [30, 15]. Several recent work propose to leverage deep architecture [6, 36, 9] and learn graph similarities at the representation level. In this work, instead of proposing yet another deep architecture for graph classification, we provide insights for the representation power of GCNs using well-known generation models. Our insights can provide guidance for choosing similarity measures in graph classification.

6 Graph Stethoscope: Distinguishing Graph Generation Models

An important application of learning graph moments is to distinguish different random graph generation models. For random graph generation processes like the BA model, the asymptotic behavior ($N \to \infty$) is known, such as scale-free. However, when the number of nodes is small, it is generally difficult to distinguish collections of graphs with different graph topology if the generation process is random. Thus, building an efficient tool that can probe the structure of small graphs of $N < 50$ like a stethoscope can be highly challenging, especially when all the graphs have the same number of nodes and edges.

BA vs. ER. We consider two tasks for graph stethoscope. In the first setting, we generate $5,000$ graphs with the same number of nodes and vary the number of edges, half of which are from the Barabasi-Albert (BA) model and the other half from the Erdos-Renyi (ER) model. In the BA model, a new node attaches to m existing nodes with a likelihood proportional to the degree of the existing nodes. The $2,500$ BA graphs are evenly split with $m = 1, N/8, N/4, 3N/8, N/2$. To avoid the bias from the order of appearance of nodes caused by preferential attachment, we shuffle the node labels. ER graphs are random undirected graphs with a probability p for generating every edge. We choose four values for p uniformly between $1/N$ and $N/2$. All graphs have similar number of edges.

BA vs. Configuration Model One might argue that distinguishing BA from ER for small graphs is easy as BA graphs are known to have a power-law distribution for the node degrees [1], and ER graphs have a Poisson degree distribution. Hence, we create a much harder task where we compare BA graphs with "fake" BA graphs where the nodes have the same degree but all edges are rewired using the Configuration Model [26] (Config.). The resulting graphs share exactly the same degree distribution. We also find that higher graph moments of the Config BA are difficult to distinguish from real BA, despite the Config. model not fixing these moments. Distinguishing BA and Config BA is very difficult using standard methods such as a Kolmogorov-Smirnov (KS) test. KS test measures the distributional differences of a statistical measure between two graphs and uses hypothesis testing to identify the graph generation model. Figure 5 shows the KS test values for pairs of real-real BA (blue) and pairs of real-fake BA (orange) w.r.t different graph moments. The dashed black lines show the mean of the KS test values for real-real pairs. We observe that the distributions of differences in real-real pairs are almost the same as those of real-fake pairs, meaning the variability in different graph moments among real BA graphs is almost the same as that between real and Config BA graphs.

Table 1: Test accuracy with different modules combinations for BA-ER. $f_1 = A$, $f_2 = D^{-1}A$, and $f_3 = D^{-1/2}AD^{-1/2}$.

Modules	`Accuracy`
f_1	53.5 %
f_3	76.9 %
f_1, f_3	89.4 %
f_1, f_2, f_3	98.8 %

Classification Using our GCN Module We evaluate the classification accuracy for these two settings using the modular GCN design, and analyze the trends of representation power w.r.t network depth and width, as well as the number of nodes in the graph. Our architecture consists of layers of our GCN module (Fig. 3, linear activation). The output is passed to a fully connected layer with softmax activation, yielding and $N \times c$ matrix (N nodes in graph, c label classes). The final

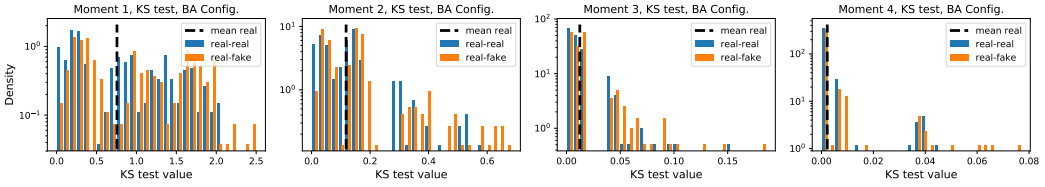

Figure 5: Distribution of Kolmogorov-Smirnov (KS) test values for differences between graph the first four graph moments $\sum_i (A^p)_{ij}$ in the dataset. "real-real" shows the distribution of KS test when comparing the graph moments of two real instances of the BA. All graphs have $N = 30$ nodes, but varying number of links. The "real-fake" case does the KS test for one real BA against one fake BA created using the configuration model.

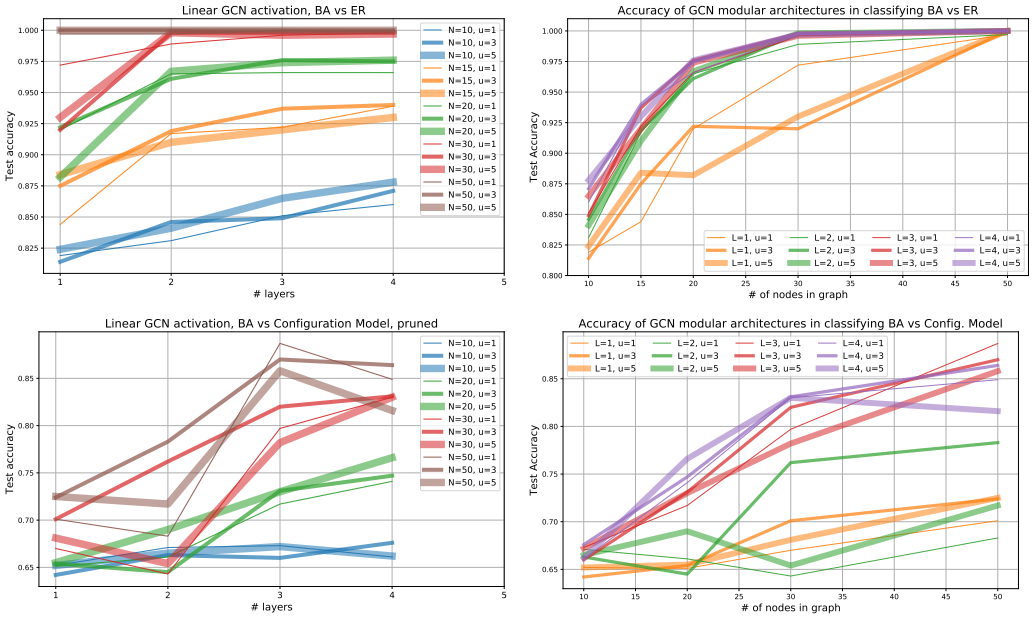

Figure 6: Classify graphs of **Barabasi-Albert** model vs. **Erdos-Renyi** model (top) and **Barabasi-Albert** model vs. **configuration** model (bottom). Left: test accuracy with respect to network depth for different number of nodes (N) and number of units (U). Right: test accuracy with respect to graph size for different number of layers (L) and number of units (U).

classification is found by mean-pooling over the N outputs. We used mean-pooling to aggregate node-level representations, after which a single number is passed to a classification layer. Figure 6 left column shows the accuracy with increasing number of layers for different number of layers and hidden units. We find that depth is more influential than width: increasing one layer can improve the test accuracy by at least 5%, whereas increasing the width has very little effect. The right column is an alternative view with increasing size of the graphs. It is clear that smaller networks are harder to learn, while for $N \geq 50$ nodes is enough for 100% accuracy in BA-ER case. BA-Config is a much harder task, with the highest accuracy of 90%.

We also conduct ablation study for our modular GCN design. Table 1 shows the change of test accuracy when we use different combinations of modules. Note that the number of parameters are kept the same for all different design. We can see that a single module is not enough to distinguish graph generation models with an accuracy close to random guessing. Having all three modules with different propagation rules leads to almost perfect discrimination between BA and ER graphs. This demonstrates the benefits of combining GCN modules to improve its representation power.

7 Conclusion

We conduct a thorough investigation in understanding what can/cannot be learned by GCNs. We focus on graph moments, a key characteristic of graph topology. We found that GCNs are rather restrictive in learning graph moments, and multi-layer GCNs cannot learn graph moments even with nonlinear activation. Theoretical analysis suggests a modular approach in designing graph neural networks while preserving permutation invariance. Modular GCNs are capable of distinguishing different graph generative models for surprisingly small graphs. Our investigation suggests that, for learning graph moments, depth is much more influential than width. Deeper GCNs are more capable of learning higher order graph moments. Our experiments also highlight the importance of combining GCN modules with residual connections in improving the representation power of GCNs.

Acknowledgments

This work was supported in part by NSF #185034, ONR-OTA (N00014-18-9-0001).

References

[1] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

[2] Andrew R Barron. Approximation and estimation bounds for artificial neural networks. *Machine learning*, 14(1):115–133, 1994.

[3] Peter L Bartlett. The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. *IEEE transactions on Information Theory*, 44(2):525–536, 1998.

[4] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[5] John Adrian Bondy and Uppaluri Siva Ramachandra Murty. Graph theory, volume 244 of. *Graduate texts in Mathematics*, 2008.

[6] Stephen Bonner, John Brennan, Georgios Theodoropoulos, Ibad Kureshi, and Andrew Stephen McGough. Deep topology classification: A new approach for massive graph classification. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 3290–3297. IEEE, 2016.

[7] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[8] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[9] James P Canning, Emma E Ingram, Sammantha Nowak-Wolff, Adriana M Ortiz, Nesreen K Ahmed, Ryan A Rossi, Karl RB Schmitt, and Sucheta Soundarajan. Predicting graph categories from structural properties. *arXiv preprint arXiv:1805.02682*, 2018.

[10] Simon S Du, Kangcheng Hou, Barnabás Póczos, Ruslan Salakhutdinov, Ruosong Wang, and Keyulu Xu. Graph neural tangent kernel: Fusing graph neural networks with graph kernels. *arXiv preprint arXiv:1905.13192*, 2019.

[11] Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. In *Conference on learning theory*, pages 907–940, 2016.

[12] Steven Farrell, Paolo Calafiura, Mayur Mudigonda, Dustin Anderson, Jean-Roch Vlimant, Stephan Zheng, Josh Bendavid, Maria Spiropulu, Giuseppe Cerati, Lindsey Gray, et al. Novel deep learning methods for track reconstruction. *arXiv preprint arXiv:1810.06111*, 2018.

[13] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org, 2017.

[14] Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018.

[15] Ting Guo and Xingquan Zhu. Understanding the roles of sub-graph features for graph classification: an empirical study perspective. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 817–822. ACM, 2013.

[16] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

[17] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[19] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[20] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[21] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *International Conference on Learning Representations (ICLR)*, 2018.

[22] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[23] Yaw-Ling Lin and Steven S Skiena. Algorithms for square roots of graphs. *SIAM Journal on Discrete Mathematics*, 8(1):99–118, 1995.

[24] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019.

[25] Ryan Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Relational pooling for graph representations. In *International Conference on Machine Learning*, pages 4663–4673, 2019.

[26] Mark Newman. *Networks: an introduction*. Oxford university press, 2010.

[27] Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl Dickstein. On the expressive power of deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2847–2854. JMLR. org, 2017.

[28] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[29] Matus Telgarsky. Benefits of depth in neural networks. *arXiv preprint arXiv:1602.04485*, 2016.

[30] Johan Ugander, Lars Backstrom, and Jon Kleinberg. Subgraph frequencies: Mapping the empirical and extremal geography of large graph collections. In *Proceedings of the 22nd international conference on World Wide Web*, pages 1307–1318. ACM, 2013.

[31] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[32] Saurabh Verma and Zhi-Li Zhang. Stability and generalization of graph convolutional neural networks. *arXiv preprint arXiv:1905.01004*, 2019.

[33] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.

[34] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[35] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. *arXiv preprint arXiv:1806.03536*, 2018.

[36] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1365–1374. ACM, 2015.

[37] Jiaxuan You, Bowen Liu, Rex Ying, Vijay Pande, and Jure Leskovec. Graph convolutional policy network for goal-directed molecular graph generation. *arXiv preprint arXiv:1806.02473*, 2018.

[38] Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. *arXiv preprint arXiv:1812.04202*, 2018.

A Learning on graphs using single hidden layer fully connected network

Proof of Theorem 1. A fully connected neural network with sigmoid activation function f_n, in principle, could approximate any function f, provided there is enough training data. Barron's result [2] states that the upper-bound on the approximation error of a single layer is given by

$$\epsilon := \|f - f_n\| \sim O\left(\frac{C_f^2}{n}\right) + O\left(\frac{nd}{S}\log S\right), \tag{11}$$

where n is the number of neurons, d is the dimension of the input, S is the number of samples, and C_f is the L_1 norm of the Fourier coefficients of the function f.

$$\begin{aligned} C_f &= \int d^d w |w|_1 |\tilde{f}(w)| \\ f(x) &= \int d^d w e^{iwx} \tilde{f}(w) \\ |w|_1 &= \sum_{j=1}^{d} |w_j|. \end{aligned} \tag{12}$$

Using (11), we can bound the approximation error of for learning graph moments. Assume that the input is a graph with N nodes, represented by an adjacency matrix A, the dimension of the input is thus $d = N^2$. If the number of nodes is not too large ($\log N \sim O(1)$), the second term in (11) essentially states that we need $S \sim O(N^2)$ samples to approximate any function well and avoid overfitting. The first term in (11) depends on the form of the function f. Specifically, C_f depends on the Fourier coefficients which have a non-negligible magnitude. Consider for example a polynomial function $f(x) = \sum_{k=0}^{p} c_k x^k$ of order p of a single variable x defined over the unit interval $I = [-1, 1]$, so that it is Lebesgue integrable. Performing the Fourier transform over this interval yields a Fourier series with coefficients given by

$$\begin{aligned} f(x) &= \sum_{k=0}^{p} c_k x^k = \sum_{m=0} \tilde{f}(m) e^{-2\pi i m x} \\ \tilde{f}(m) &= \sum_{k=0} c_k \int_{-1}^{1} \frac{dx}{2\pi i} x^j e^{-2\pi i m x} = \frac{c_k}{2\pi i k!} \frac{\partial^k}{\partial m^k} \delta(m). \end{aligned} \tag{13}$$

If all the coefficients $c_k \sim O(1)$, (13) states that at most p Fourier coefficients will have $O(1)$ magnitudes for this polynomial function and so $C_f \sim O(p)$ for a polynomial f of a single variable x. If x is d dimensional, we have $C_f \sim O(pd)$. We want to learn graph moments, which are polynomial functions of the elements in A_{ij}. For example, the node degree is a first order polynomial of the form $f(A)_i = \sum_{j=0}^{N} A_{ij}$. Higher order moments are generally functions of higher powers of A. For example, the number paths of length two between nodes nodes i and j on an unweighted graph are given by $P_{ij} = \sum_k A_{ik} A_{kj}$. We can write this as a second order function in A_{ij}

$$P_{ij} = \sum_{k,l=1}^{N} A_{ik} A_{lj} c^{kl}$$

In general, for a graph moment of order p, denoted as $M_p(A)$, we have an expression:

$$M_p(A) = \prod_{q=0}^{p} c^{i_q k_q} A_{k_q j_{q+1}}. \tag{14}$$

which could have as many as $O(pN^2)$ or at least $O(pN)$ nonzero coefficients. Assuming all these nonzero coefficients are $O(1)$, we get $C_f \sim O(pN^q)$ with $1 \leq q \leq 2$. Thus, in order for the first term in (11) to be small, we need $n > O(C_f^2) \sim O(p^2 N^{2q})$ neurons in the best case, or $n > O(p^2 N^4)$ in the worst case. Additionally, to make the second error term in (11) small, we would need $S > O(nd) \sim O\left(p^2 N^{2q+2}\right)$ samples. □

For many real world graphs, we have relatively few samples (S) and a large number of nodes (N), using a fully-connected network for learning network moments is nearly impossible. However, note that graph moments are invariant under node permutations. Similar to how Convolutional Neural Networks (CNNs) exploit translation invariance to drastically reduce the number of parameters needed to learn spatial features, graph convolutional networks (GCNs) exploit node permutation invariance, constraining the weights to be the same for all nodes. Additionally, the weights can also not treat neighbors of nodes differently, as neighbors can be permuted too.

The restriction of being permutation invariant also reduces the representation power of a GCN, forcing it to take a very simple form. Namely, the weights of a GCN w^a are simply multiplied into all entries of A_{ij}. This architecture is node permutation invariant, but it also uses node attributes to couple the weights to neighborhoods of nodes. Denote h_i^a as the attribute a of node i. The output of a GCN follows the formula below:

$$F(A,h)_i^\mu = \sigma\left(\sum_j A_{ij} h_j^a W_a^\mu + b^\mu\right) \tag{15}$$

where μ denotes the output dimension and b is the bias term. In principle, A_{ij} can be replaced by any general function $f(A)_{ij}$, defined by the propagation rule.

Following the reasoning above, learning nonlinear functions for $F(A)$ requires a lot of data and parameters. It is, therefore, much easier to combine different propagation rules, aka modules related to the generation processes of the graph, such as diffusion operators $D^{-1}A$ and $D^{-1/2}AD^{-1/2}$ and use them instead of only A. We also add a node-wise dense layer (which act similar to a GCN, not mixing different nodes) after each of these operators to mix the outputs of these operators.

B Experiment details

we generate $5,000$ graphs with the same number of nodes and varying number of links, half of which are from the Barabasi-Albert (BA) model and the other half from the Erdos-Renyi (ER) model. In the BA model, new nodes attach to m existing nodes with likelihood p_i proportional to the degree of the existing node i.

$$p_i = \frac{d_i}{\sum_i d_i}$$

The $2,500$ BA graphs are evenly split with $m = 1, N/8, N/4, 3N/8, N/2$. To avoid bias from order of appearance of nodes caused by preferential attachment, we shuffle the node labels. ER graphs are random undirected graphs with a probability p for every link. We choose four values for p uniformly between $1/N$ and $N/2$. All graphs have similar number of links.

For a configuration model [26], the links are generated as follows: Take a degree sequence, i. e. assign a degree d_i to each node. The degrees of the nodes are represented as half-links or stubs. The sum of stubs must be even in order to be able to construct a graph ($\Sigma d_i = 2m$). The degree sequence is drawn from the adjacency matrix of the BA graph. Choose two nodes uniformly at random. Connect them with an edge using up one of each node's stubs. Choose another pair from the remaining $2m - 2$ stubs and connect them. Continue until running out of stubs. The result is a graph with the pre-defined degree sequence. We rewire the edges of BA graphs to obtain "fake" BA graphs. The resulting graphs share exactly the same degree distribution, and even mimic the real BA in higher graph moments.

C Learning graph moments without residual connections

Our first attempt to combine different GCN modules is to stack them on top of each other in a feed-forward way mimicking multi-layer GCNs. However, our theoretical analysis shows the limited representation power of this design. In particular, no matter how many layers or how non-linear the activation function gets, multiple GCN layers stacked in a feed-forward way cannot perform well in learning network moments whose order is not precisely the number of layers. We observe in our experiments that this is indeed the case.

As shown in Fig 7 shows the test loss over number of epochs for learning first-order (top), second-order (middle) and third-order (bottom) graph moments. We vary the number of layers from 1 to 4

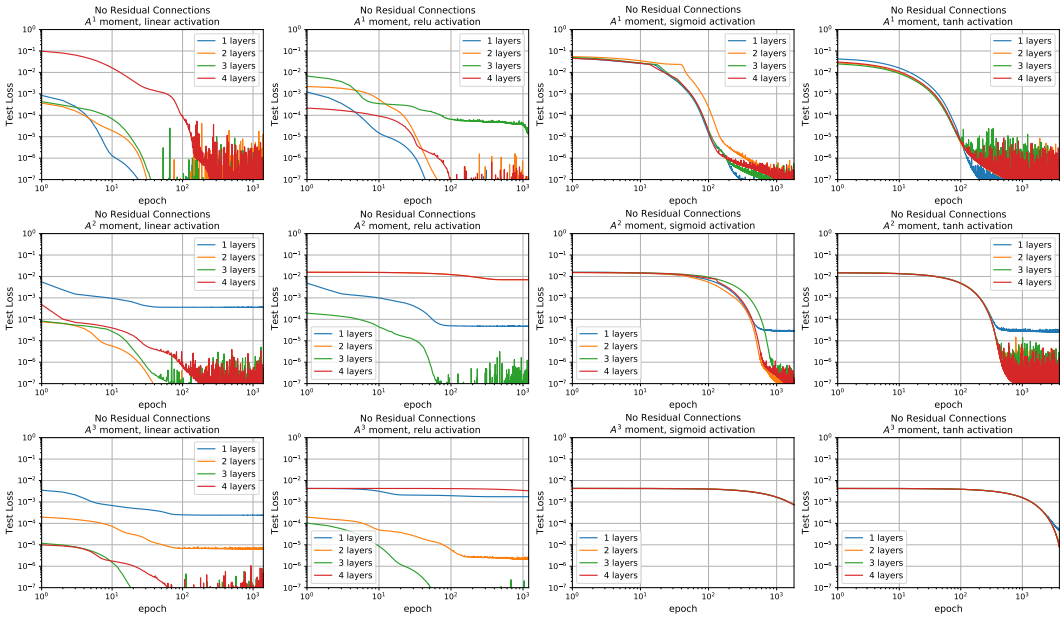

Figure 7: Expressiveness of GCN module *without* Residual Connections: learning first (top), second (middle) and third (bottom) order graph moments with multiple GCN layers and different activation functions. GCN without residual connections fails to learn well when the target graph moments order is greater than the number of layers. With ReLU, sometimes more layers performs even worse. Also, without residuals higher number of layers doesn't always perform as good as when the number of layer matches the order of the moment exactly.

and test with different activation functions including linear, ReLU, sigmoid and tanh. GCN without residual connections fails to learn well when the target graph moments order is greater than the number of layers. With ReLU, sometimes more layers performs even worse. Also, without residuals higher number of layers doesn't always perform as good as when the number of layer matches the order of the moment exactly.

D A Note on Graph Attention Networks

The Graph Attention Network (GAT) [31] modifies a message-passing neural network such as GCN by modifying the weights on the edges of the graph. This changes how much each neighbor j of a node i plays a role in the output of node i. Assume the input of the GCN layer where the attention layer is added has F features per node, and the output has F' features. Take the $F \times F'$ shared weight matrix W of GCN. Graph attention uses a function $a : \mathbb{R}^{F'} \times \mathbb{R}^{F'} : \mathbb{R}$ to look at similarities of the linear outputs for different nodes

$$e_{ij} = a(Wh_i, Wh_j) \tag{16}$$

However, e_{ij} is only computed for neighbors, meaning that what we really calculate is

$$e_{ij} = a(Wh_i, Wh_j) \circ \hat{A}_{ij} \tag{17}$$

where $\circ$ denotes element-wise multiplication and $\hat{A}$ is the unweighted adjacency matrix of the graph. The function a is implemented as a neural network with a $2F' \times 1$ weight matrix and softmax activation. Now, consider an unweighted graph so that $\hat{A} = A$. The attention mechanism is a function over neighbors only. It takes the output features and passes a linear combination through an activation function σ (usually Leaky ReLU)

$$e_{ij} = \sigma\left(\alpha_1 \cdot W \cdot h_i + \alpha_2 \cdot W \cdot h_j\right) \tag{18}$$

where α_n are $F' \times 1$ weight matrices for the attention layer. This output is then also passed through softmax over the neighbors of node i. Compared with normal GCN, the attention network is deciding which neighbors should get more weight in the output of node i. Our modular approach does not distinguish among neighbors and is a regular message-passing neural network. We concatenate the output of multiple propagation rules, but each rule is still used in a regular message passing step.