Graph Structures, Random Walks, and All That: Learning Graphs with Jumping Knowledge Networks

by

Keyulu Xu

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2019

Author .....
Department of Electrical Engineering and Computer Science
December 14, 2018

Certified by .....
Stefanie Jegelka
Assistant Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Graph Structures, Random Walks, and All That: Learning Graphs with Jumping Knowledge Networks

by

Keyulu Xu

Submitted to the Department of Electrical Engineering and Computer Science
on December 14, 2018, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

Graph representation learning aims to extract high-level features from the graph structures and node features, in order to make predictions about the nodes and the graphs. Applications include predicting chemical properties of drugs, community detection in social networks, and modeling interactions in physical systems.

Recent deep learning approaches for graph representation learning, namely Graph Neural Networks (GNNs), follow a neighborhood aggregation procedure, where the representation vector of a node is computed by recursively aggregating and transforming feature vectors of its neighboring nodes. We analyze some important properties of these models, and propose a strategy to overcome the limitations. In particular, the range of neighboring nodes that a node's representation draws from strongly depends on the graph structure, analogous to the spread of a random walk. To adapt to local neighborhood properties and tasks, we explore an architecture – jumping knowledge (JK) networks that flexibly leverages, for each node, different neighborhood ranges to enable better structure-aware representation.

In a number of experiments on social, bioinformatics and citation networks, we demonstrate that our model achieves state-of-the-art performance. Furthermore, combining the JK framework with models like Graph Convolutional Networks, GraphSAGE and Graph Attention Networks consistently improves those models' performance.

Thesis Supervisor: Stefanie Jegelka
Title: Assistant Professor of Electrical Engineering and Computer Science

Acknowledgments

First and foremost, I would like to thank my research advisor, Stefanie Jegelka for her encouraging conversations, insightful guidance, and visionary suggestions. Without her extremely helpful support and advice, this thesis could have never been complete.

I would also like to express my gratitude to my undergraduate advisor Nick Harvey, for introducing me to the field of research in learning theory and algorithms, and encouraging me to pursue my graduate study at MIT.

I would like to thank my friends and colleagues Chengtao Li, Yonglong Tian, and Tomohiro Sonobe for their help on research, especially on the experiment and parameter tuning components of this thesis. I would like to thank Prof. Ken-ichi Kawarabayashi for his help on the graph theory component of this thesis as well as encouraging comments.

Thank you to my officemates, Matt J. Staib and Chengtao Li for providing a supportive and relaxed environment for research and conversations. Especially thanks to Matt J. Staib and Chengtao Li for their encouraging and supportive conversations during the hardest time of school and research.

Thank you to my groupmate Zi Wang for her positive and encouraging comments during the hardest time of research and life, as well as visionary conversations on research.

Last but not the least, I would like to thank my family and friends, who have been very supportive during the hard time of life and research. I would like to dedicate this thesis to them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview and Motivation

Graphs are a ubiquitous structure that widely occurs in data analysis problems. Real-world graphs such as social networks, financial networks, biological networks and citation networks represent important rich information which is not seen from the individual entities alone, for example, the communities a person is in, the functional role of a molecule, and the sensitivity of the assets of an enterprise to external shocks.

Representation learning of nodes in graphs aims to extract high-level features from a node as well as its neighborhood, and has proved extremely useful for many applications, such as node classification, clustering, and link prediction [31, 30, 10, 37]. Representation learning of graphs leverage the node representations and the graph structures to obtain an embedding of the entire graph. Applications include graph classification and graph generation [42, 43].

Recent works focus on deep learning approaches to graph representation learning, which are generally named Graph Neural Networks (GNNs). Many GNN variants have been proposed and have achieved state-of-the-art results on both node and graph classification tasks [26, 11, 22, 39]. Graph Neural Network (GNN) approaches broadly follow a neighborhood aggregation (or "message passing" scheme), and those have been very promising [33, 3, 7, 8, 11, 20, 22, 26, 39, 40, 43, 44]. These models learn to iteratively aggregate the hidden features of every node in the graph with its adjacent

nodes' as its new hidden features, where an iteration is parametrized by a layer of the neural network. After k iterations of aggregation, a node is represented by its transformed feature vector, which captures the structural information within the node's k-hop network neighborhood. The representation of an entire graph can then be obtained through pooling, for example, by summing the representation vectors of all nodes in the graph. Theoretically, an aggregation process of k iterations makes use of the subtree structures of height k rooted at every node. Such schemes have been shown to generalize the Weisfeiler-Lehman graph isomorphism test [41] enabling to simultaneously learn the topology as well as the distribution of node features in the neighborhood [35, 22, 11].

Yet, such aggregation schemes sometimes lead to surprises. For example, it has been observed that the best performance with one of the state-of-the-art GNN models, Graph Convolutional Networks (GCN), is achieved with a 2-layer model. Deeper versions of the model that, in principle, have access to more information, perform worse [22]. A similar degradation of learning for computer vision problems is resolved by residual connections [13] that greatly aid the training of deep models. But, even with residual connections, GCNs with more layers do not perform as well as the 2-layer GCN on many datasets, e.g. citation networks [22].

Here are some natural follow-up questions one may ask. 1) What leads to such "curse of depth" in Graph Neural Networks? 2) Why does the residual connection not resolve the problem for graph learning as in computer vision? 3) Are models with 2-layer really the best, or we can do better?

Motivated by observations and questions like the above, in this thesis, we address two questions. First, we study properties and resulting limitations of the neighborhood aggregation schemes in Graph Neural Networks. Our analysis relates deep learning architectures to the graph structures and random walk distributions. In particular, our theoretical results of GNNs will answer the three questions raised above and reveal a promising direction for deeper while better models. Second, based on this analysis, we propose an architecture that, as opposed to existing models, enables adaptive, *structure-aware* representations to overcome the "curse of depth" degration above.

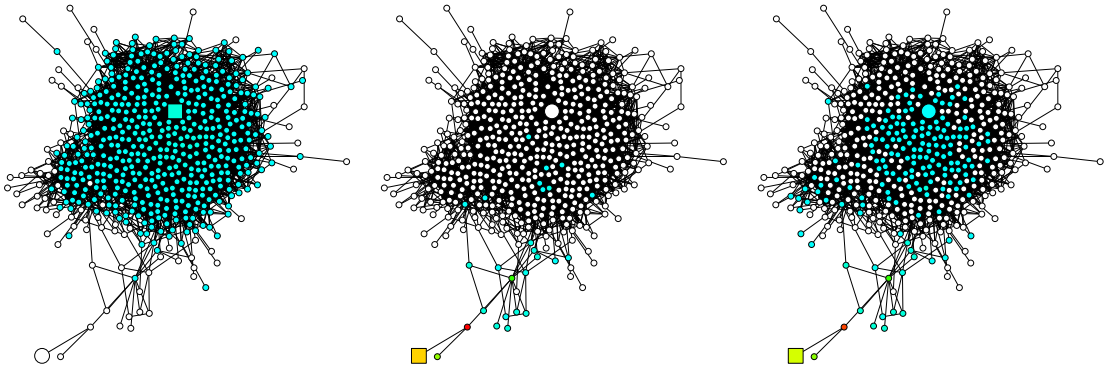
Such representations are particularly interesting for representation learning on large complex graphs with diverse subgraph structures.

Model analysis. To better understand the behavior of different neighborhood aggregation schemes, we analyze the effective range of nodes that any given node's representation draws from. We summarize this sensitivity analysis by what we name the *influence distribution* of a node. This effective range implicitly encodes prior assumptions on what are the "nearest neighbors" that a node should draw information from. In particular, we will see that this influence is heavily affected by the graph structure, raising the question whether "one size fits all", in particular in graphs whose subgraphs have varying properties (such as more tree-like or more expander-like).

In particular, our more formal analysis connects influence distributions with the spread of a random walk at a given node, a well-understood phenomenon as a function of the graph structure and eigenvalues [27]. For instance, in some cases and applications, a 2-step random walk influence that focuses on local neighborhoods can be more informative than higher-order features where some of the information may be "washed out" via averaging.

Changing locality. To illustrate the effect and importance of graph structure, recall that many real-world graphs possess locally strongly varying structure. In biological and citation networks, the majority of the nodes have few connections, whereas some nodes (hubs) are connected to many other nodes. Social and web networks usually consist of an expander-like core part and an almost-tree (bounded treewidth) part, which represent well-connected entities and the small communities respectively [24, 29, 38].

Besides node features, this subgraph structure has great impact on the result of neighborhood aggregation. The speed of expansion or, equivalently, growth of the influence radius, is characterized by the random walk's mixing time, which changes dramatically on subgraphs with different structures [27]. Thus, the same number of iterations (layers) can lead to influence distributions of very different locality. As an example, consider the social network in Figure 1-1 from GooglePlus [25]. The figure illustrates the expansions of a random walk starting at the square node. The



(a) 4 steps at core (b) 4 steps at tree (c) 5 steps at tree

Figure 1-1: Expansion of a random walk (and hence influence distribution) starting at (square) nodes in subgraphs with different structures. Different subgraph structures result in very different neighborhood sizes.

walk (a) from a node within the core rapidly includes almost the entire graph. In contrast, the walk (b) starting at a node in the tree part includes only a very small fraction of all nodes. After 5 steps, the same walk has reached the core and, suddenly, spreads quickly. Translated to graph representation models, these spreads become the influence distributions or, in other words, the averaged features yield the new feature of the walk's starting node. This shows that in the same graph, the same number of steps can lead to very different effects. Depending on the application, wide-range or small-range feature combinations may be more desirable. A too rapid expansion may average too broadly and thereby lose information, while in other parts of the graph, a sufficient neighborhood may be needed for stabilizing predictions.

JK networks. The above observations raise the question whether it is possible to adaptively *adjust* (i.e., learn) the influence radii for each node and task. To achieve this, we explore an architecture that learns to selectively exploit information from neighborhoods of differing locality. This architecture selectively combines different aggregations at the last layer, i.e., the representations "jump" to the last layer. Hence, we name the resulting networks *Jumping Knowledge Networks (JK-Nets)*. We will see that empirically, when adaptation is an option, the networks indeed learn representations of different orders for different graph substructures. Moreover, in Section 4.1, we show that applying our framework to various state-of-the-art neighborhood-aggregation

models consistently improves their performance.

1.2 Background and Graph Neural Networks

We begin by summarizing some of the most common Graph Neural Network (GNN) models and their neighborhood aggregation schemes. Along the way, we introduce our notation.

Let $G = (V, E)$ be a simple graph with node features $X_v \in \mathbb{R}^{d_i}$ for $v \in V$. Let $\widetilde{G}$ be the graph obtained by adding a self-loop to every $v \in V$. The hidden feature of node v learned by the l-th layer of the model is denoted by $h_v^{(l)} \in \mathbb{R}^{d_h}$. Here, d_i is the dimension of the input features and d_h is the dimension of the hidden features, which, for simplicity of exposition, we assume to be the same across layers. We also use $h_v^{(0)} = X_v$ for the node feature. The neighborhood $N(v) = \{u \in V | \, (v, u) \in E\}$ of node v is the set of adjacent nodes of v. The analogous neighborhood $\widetilde{N}(v) = \{v\} \cup \{u \in V | \, (v, u) \in E\}$ on $\widetilde{G}$ includes v.

Graph Neural Networks use the graph structure and node features X_v to learn a representation vector of a node, h_v, or the entire graph, h_G. Modern GNNs follow a neighborhood aggregation strategy, where we iteratively update the representation of a node by aggregating representations of its neighbors. After k iterations of aggregation, a node's representation captures the structural information within its k-hop network neighborhood. Formally, the k-th layer of a GNN can generally be formulated as

$$a_v^{(k)} = \text{AGGREGATE}^{(k)} \left(\left\{ h_u^{(k-1)} : u \in \mathcal{N}(v) \right\} \right), \tag{1.1}$$

$$h_v^{(k)} = \text{COMBINE}^{(k)} \left(h_v^{(k-1)}, a_v^{(k)} \right), \tag{1.2}$$

where $h_v^{(k)}$ is the feature vector of node v at the k-th iteration/layer. We initialize $h_v^{(0)} = X_v$, and $\mathcal{N}(v)$ is a set of nodes adjacent to v.

The specific choice of $\text{AGGREGATE}^{(k)}(\cdot)$ and $\text{COMBINE}^{(k)}(\cdot)$ in GNNs are crucial. A number of architectures for AGGREGATE and COMBINE have been proposed. Next, we introduce some of the most common GNN variants and their concrete

neighborhood aggregation functions formulated in the framework above.

Graph Convolutional Networks (GCN). Graph Convolutional Networks (GCN) [22], initially motivated by spectral graph convolutions [12, 7], are a specific instantiation of this framework [9], of the form

$$h_v^{(l)} = \mathrm{ReLU}\Big(W_l \cdot \sum_{u \in \widetilde{N}(v)} (\deg(v)\deg(u))^{-1/2}\, h_u^{(l-1)}\Big) \tag{1.3}$$

where deg(v) stands for the degree of node v in G. Here, W_l is a learnable feature transform matrix at layer l, and it is shared among all nodes. [11] derived a variant of GCN that also works in inductive settings (previously unseen nodes), by using a different normalization to average:

$$h_v^{(l)} = \mathrm{ReLU}\Big(W_l \cdot \frac{1}{\widetilde{\deg}(v)} \sum_{u \in \widetilde{N}(v)} h_u^{(l-1)}\Big) \tag{1.4}$$

where $\widetilde{\deg}(v)$ is the degree of node v in $\widetilde{G}$. Note that in Graph Convolutional Networks (GCN), the AGGREGATE and COMBINE steps are integrated by aggregating the node along with its neighbors.

Neighborhood Aggregation with Skip Connections. Instead of aggregating a node and its neighbors at the same time as in GCN in Eqn. (1.4), a number of recent approaches aggregate the neighbors first and then combine the resulting neighborhood representation with the node's representation from the last iteration. More formally, each node is updated as

$$\begin{aligned} h_{N(v)}^{(l)} &= \sigma\left(W_l \cdot \mathrm{AGGREGATE}_N\big(\{h_u^{(l-1)}, \forall u \in N(v)\}\big)\right) \\ h_v^{(l)} &= \mathrm{COMBINE}\left(h_v^{(l-1)}, h_{N(v)}^{(l)}\right) \end{aligned}$$

where $\mathrm{AGGREGATE}_N$ and COMBINE are defined by the specific model. The COMBINE step is key to this paradigm and can be viewed as a form of a "skip

connection" between different layers. For COMBINE, a GNN variant GraphSAGE [11] uses concatenation after a feature transform, and the element-wise mean operation in (1.4) is replaced by max-pooling or LSTM functions. More specifically, the pooling version of GraphSAGE is formulated as follows.

$$a_v^{(k)} = \text{MAX}\left(\left\{\text{ReLU}\left(W \cdot h_u^{(k-1)}\right),\ \forall u \in \mathcal{N}(v)\right\}\right), \tag{1.5}$$

$$h_v^{(k)} = W \cdot \left[h_v^{(k-1)} \middle| a_v^{(k)}\right] \tag{1.6}$$

where W are learnable feature transform parameters, and the brackets stand for concatenation. There have also been other schemes for COMBINE. Column Networks [32] interpolate the neighborhood representation and the node's previous representation, and Gated GNN [26] uses the Gated Recurrent Unit (GRU) [5]. Another well-known variant of skip connections, residual connections, use the identity mapping to help signals propagate [13, 14].

These skip connections are input- but not output-unit specific: If we "skip" a layer for $h_v^{(l)}$ (do not aggregate) or use a certain COMBINE, all subsequent units using this representation will be using this skip implicitly. It is impossible that a certain higher-up representation $h_u^{(l+j)}$ uses the skip and another one does not. As a result, skip connections cannot adaptively adjust the neighborhood sizes of the final-layer representations independently. Later we will show that due to reasons like the above, skip connections cannot fundamentally resolve the "curse of depth" problem raised before.

Neighborhood Aggregation with Directional Biases. Some recent models, rather than treating the features of adjacent nodes equally, weigh "important" neighbors more. This paradigm can be viewed as neighborhood-aggregation with directional biases because a node will be influenced by some directions of expansion more than the others.

Graph Attention Networks (GAT) [39] and VAIN [17] learn to select the important neighbors via an attention mechanism. The max-pooling operation in GraphSAGE [11]

implicitly selects the important nodes. This line of work is orthogonal to ours, because it modifies the direction of expansion whereas our model operates on the locality of expansion. Our model can be combined with these models to add representational power. In Section 4.1, we demonstrate that our framework works with not only simple neighborhood-aggregation models (GCN), but also with skip connections (GraphSAGE) and directional biases (GAT).

Graph-level Readout. For node classification, the node representation $h_v^{(K)}$ of the final iteration is used for prediction. For representation learning of an entire graph and tasks like graph classification, the READOUT function aggregates node features from the final iteration to obtain the entire graph's representation h_G:

$$h_G = \text{READOUT}\big(\{h_v^{(K)} \mid v \in G\}\big). \tag{1.7}$$

READOUT can be a simple permutation invariant function such as summation or a more sophisticated graph-level pooling function [43, 44].

Generally, the theoretical results and proposed frameworks in this thesis can be applied to both node and graph representation learning. For simplicity of illustration, throughout the remaining sections of the thesis, we will focus on the task of node classification.

1.3 Basic Structural Graph Theory

In this section, we introduce a few common graph structures in graph theory, which play a critical role in the effect of neighborhood aggregation procedures and thus are important in our theoretical analysis.

Expanders. The first class of graphs we introduce is *expanders*. Intuitively, an expander is a possibly *sparse* graph that has strong connectivity properties, which can be quantified via vertex, edge or spectral expansion. A graph is an expander if it has high expansion parameters. We formally define the edge expansion as follows.

Definition 1.3.1 (Edge expansion)**.** *The edge expansion (or Cheeger constant) $h(G)$ of a graph G with n vertices is defined as*

$$h(G) = \min_{0<|S|\leq \frac{2}{n}} \frac{|\partial S|}{|S|},$$

where the edge boundary $\partial S = \{(u, v) \in E(G) : u \in S, v \in V(G) \setminus S\}$

Intuitively, the higher the expansion parameter, the larger the smallest graph cut is. Thus, the graph is highly connected. When we start a random walk on an expander, due to the highly connected structure, the random walk expands very rapidly and will cover almost the entire graph in a few iterations.

Some examples of expander graphs are random regular graphs, where each node is connected to a fixed number of neighbors randomly. In many real-world graphs, such as social networks and citation networks, there usually exist expander parts, where people or academic papers are highly connected.

Bounded treewidth graphs. An "opposite" structure of expanders are bounded treewidth graphs. Intuitively, the structure and properties of such graphs are very much similar to a tree. Trees behave much differently from expanders in many graph-theoretic aspects. First, unlike expanders, trees are fairly easy to cut. For example, partitioning a binary tree into two parts through the root results in a minimal edge expansion. Therefore, trees do not exhibit the highly-connected structure. Moreover, a random walk starting in a tree-like structure will expand slowly and mostly get "trapped" inside the tree, compared to the rapid expansion in expander graphs.

Conceptually, we have discussed the properties of bounded treewidth graphs. To formally define such graphs, we introduce the notion of treewidth and tree decomposition [1]. The treewidth of an undirected graph is a number associated with the graph. The smaller the treewidth, the more tree-like the graph is. The treewidth of a tree is exactly 2. The treewidth of a graph can be computed via a process called tree decomposition. It is known that it is NP-complete to determine whether a graph has treewidth at most a given variable k [1].

In our analysis, however, we do not need to perform tree decomposition to compute the treewidth of a graph. We will be using the fact that real-world graphs contain many bounded treewidth parts and random walk expansion on such subgraphs are very slow.

Chapter 2

Theory

In this chapter, we present a theoretical framework "influence distributions" for analyzing the properties of GNN architectures. Our analysis framework relates a GNN architecture and its learned node representations to a random walk distribution. The behavior of a random walk distribution, however, is highly related to the graph structure where it starts from, e.g., expanders or bounded treewidth graphs. Therefore, we effectively relate the effect of a GNN aggregation procedure to the graph structures it is acting on. From there, we conclude that the existing GNN models are *not structure-aware*, in the sense that always the same number of random walk steps is applied to all subgraph structures, despite having undesirable learned representations due to the discrepancy in random walk expansion speed of different structures.

2.1 Influence Distribution and Random Walks

We start by exploring some important properties of the neighborhood aggregation schemes in Graph Neural Networks. Related to ideas of sensitivity analysis and influence functions in statistics [23] that measure the influence of a training point on parameters, we study the range of nodes whose features affect a given node's representation. This range gives insight into how large a neighborhood a node is drawing information from.

We measure the sensitivity of node x to node y, or the influence of y on x, by

measuring how much a change in the input feature of y affects the representation of x in the last layer. For any node x, the *influence distribution* captures the relative influences of all other nodes.

Definition 2.1.1 (Influence score and distribution)**.** *For a simple graph $G = (V, E)$, let $h_x^{(0)}$ be the input feature and $h_x^{(k)}$ be the learned hidden feature of node $x \in V$ at the k-th (last) layer of the model. The* influence score $I(x, y)$ *of node x by any node $y \in V$ is the sum of the absolute values of the entries of the Jacobian matrix $\left[\frac{\partial h_x^{(k)}}{\partial h_y^{(0)}}\right]$. We define the* influence distribution I_x *of $x \in V$ by normalizing the influence scores: $I_x(y) = I(x, y) / \sum_z I(x, z)$, or*

$$I_x(y) = e^T \left[\frac{\partial h_x^{(k)}}{\partial h_y^{(0)}}\right] e \Big/ \left(\sum_{z \in V} e^T \left[\frac{\partial h_x^{(k)}}{\partial h_z^{(0)}}\right] e\right)$$

where e is the all-ones vector.

Later, we will see connections of influence distributions with random walks. For completeness, we also define random walk distributions.

Definition 2.1.2. *Consider a random walk on $\widetilde{G}$ starting at a node v_0; if at the t-th step we are at a node v_t, we move to any neighbor of v_t (including v_t) with equal probability. The t-*step random walk distribution *P_t of v_0 is*

$$P_t(i) = Prob(v_t = i). \tag{2.1}$$

Analogous definitions apply for random walks with non-uniform transition probabilities.

An important property of the random walk distribution is that it becomes more spread out as t increases and converges to the limit distribution if the graph is non-bipartite. The rate of convergence depends on the structure of the subgraph and can be bounded by the spectral gap (or the conductance) of the random walk's transition matrix [27].

2.1.1 Model Analysis

The influence distribution for different aggregation models and nodes can give insights into the information captured by the respective representations. The following results show that the influence distributions of common aggregation schemes are closely connected to random walk distributions. This observation hints at specific implications – strengths and weaknesses – that we will discuss.

With a randomization assumption of the ReLU activations similar to that in [19, 6], we can draw connections between GCNs and random walks:

Theorem 1. *Given a k-layer GCN with averaging as in Equation (1.4), assume that all paths in the computation graph of the model are activated with the same probability of success ρ. Then the influence distribution I_x for any node $x \in V$ is equivalent, in expectation, to the k-step random walk distribution on $\widetilde{G}$ starting at node x.*

Proof. Denote by $f_x^{(l)}$ the pre-activated feature of $h_x^{(l)}$, i.e. $\frac{1}{\widetilde{\deg}(x)} \cdot \sum_{z \in \widetilde{N}(x)} W_l h_z^{(l-1)}$, for any $l = 1..k$, we have

$$\frac{\partial h_x^{(l)}}{\partial h_y^{(0)}} = \frac{1}{\widetilde{\deg}(x)} \cdot \operatorname{diag}\left(1_{f_x^{(l)}>0}\right) \cdot W_l \cdot \sum_{z \in \widetilde{N}(x)} \frac{\partial h_z^{(l-1)}}{\partial h_y^{(0)}}$$

By chain rule, we get

$$\begin{aligned}\frac{\partial h_x^{(k)}}{\partial h_y^{(0)}} &= \sum_{p=1}^{\Psi} \left[\frac{\partial h_x^{(k)}}{\partial h_y^{(0)}}\right]_p \\ &= \sum_{p=1}^{\Psi} \prod_{l=k}^{1} \frac{1}{\widetilde{\deg}(v_p^l)} \cdot \operatorname{diag}\left(1_{f_{v_p^l}^{(l)}>0}\right) \cdot W_l\end{aligned}$$

Here, Ψ is the total number of paths $v_p^k v_p^{k-1}, ..., v_p^1, v_p^0$ of length $k+1$ from node x to node y. For any path p, v_p^k is node x, v_p^0 is node y and for $l = 1..k-1$, $v_p^{l-1} \in \widetilde{N}(v_p^l)$.

As for each path p, the derivative $\left[\frac{\partial h_x^{(k)}}{\partial h_y^{(0)}}\right]_p$ represents a directed acyclic computation graph, where the input neurons are the same as the entries of W_1, and at a layer l.

We can express an entry of the derivative as

$$\left[\frac{\partial h_x^{(k)}}{\partial h_y^{(0)}}\right]_p^{(i,j)} = \prod_{l=k}^{1} \frac{1}{\widetilde{\deg}(v_p^l)} \sum_{q=1}^{\Phi} Z_q \prod_{l=k}^{1} w_q^{(l)}$$

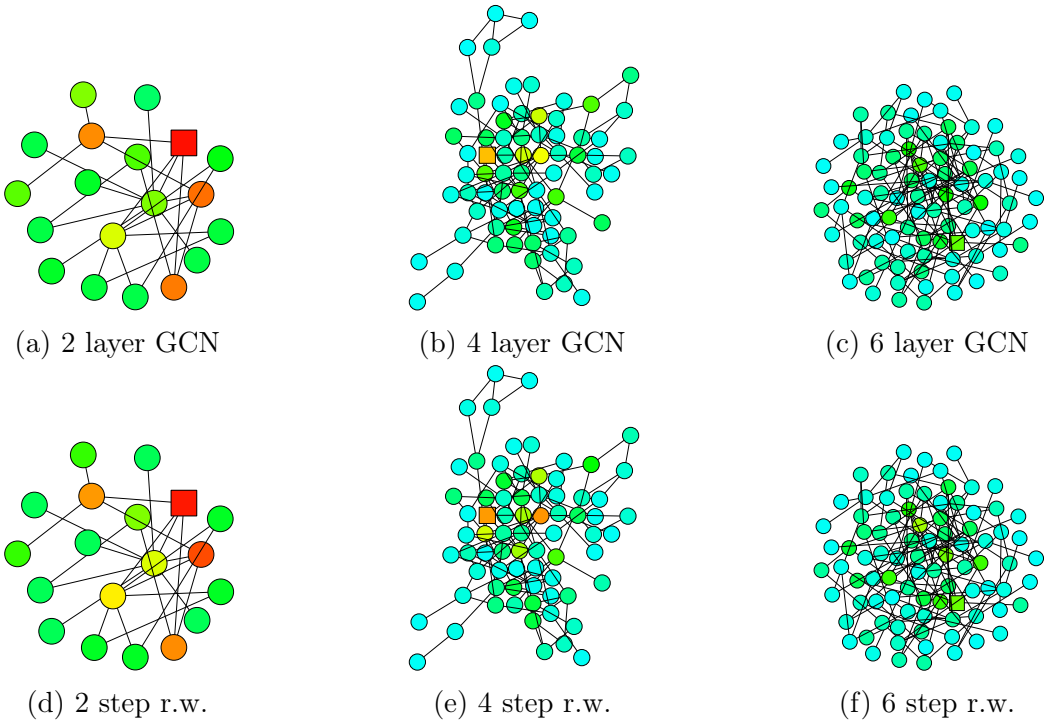
Here, Φ is the number of paths q from the input neurons to the output neuron (i, j), in the computation graph of $\left[\frac{\partial h_x^{(k)}}{\partial h_y^{(0)}}\right]_p$. For each layer l, w_q^l is the entry of W_l that is used in the q-th path. Finally, $Z_q \in \{0, 1\}$ represents whether the q-th path is active ($Z_q = 1$) or not ($Z_q = 0$) as a result of the ReLU activation of the entries of $f_{v_p^l}^{(l)}$'s on the q-th path.

Under the assumption that the Z's are Bernoulli random variables with the same probability of success, for all q, $\Pr(Z_q = 1) = \rho$, we have $\mathbb{E}\left[\left[\frac{\partial h_x^{(k)}}{\partial h_y^{(0)}}\right]_p^{(i,j)}\right] = \rho \cdot \prod_{l=k}^{1} \frac{1}{\widetilde{\deg}(v_p^l)} \cdot w_q^{(l)}$. It follows that $\mathbb{E}\left[\frac{\partial h_x^{(k)}}{\partial h_y^{(0)}}\right] = \rho \cdot \prod_{l=k}^{1} W_l \cdot \left(\sum_{p=1}^{\Psi} \prod_{l=k}^{1} \frac{1}{\widetilde{\deg}(v_p^l)}\right)$. We know that the k-step random walk probability at y can be computed by summing up the probability of all paths of length k from x to y, which is exactly $\sum_{p=1}^{\Psi} \prod_{l=k}^{1} \frac{1}{\widetilde{\deg}(v_p^l)}$. Moreover, the random walk probability starting at x to other nodes sum up to 1. We know that the influence score $I(x, z)$ for any z in expectation is thus the random walk probability of being at z from x at the k-th step, multiplied by a term that is the same for all z. Normalizing the influence scores ends the proof.

Comment: ReLU is not differentiable at 0. For simplicity, we assume the (sub)gradient to be 0 at 0. □

It is straightforward to modify the proof of Theorem 1 to show a nearly equivalent result for the version of GCN in Equation (1.3). The only difference is that each random walk path $v_p^0, v_p^1, ..., v_p^k$ from node x (v_p^0) to y (v_p^k), instead of probability $\rho \prod_{l=1}^{k} \frac{1}{\widetilde{\deg}(v_p^l)}$, now has probability $\frac{\rho}{Q} \prod_{l=1}^{k-1} \frac{1}{\widetilde{\deg}(v_p^l)} \cdot (\widetilde{\deg}(x)\widetilde{\deg}(y))^{-1/2}$, where Q is a normalizing factor. Thus, the difference in probability is small, especially when the degree of x and y are close.

Similarly, we can show that neighborhood aggregation schemes with directional biases resemble biased random walk distributions. This follows by substituting the corresponding probabilities into the proof of Theorem 1.

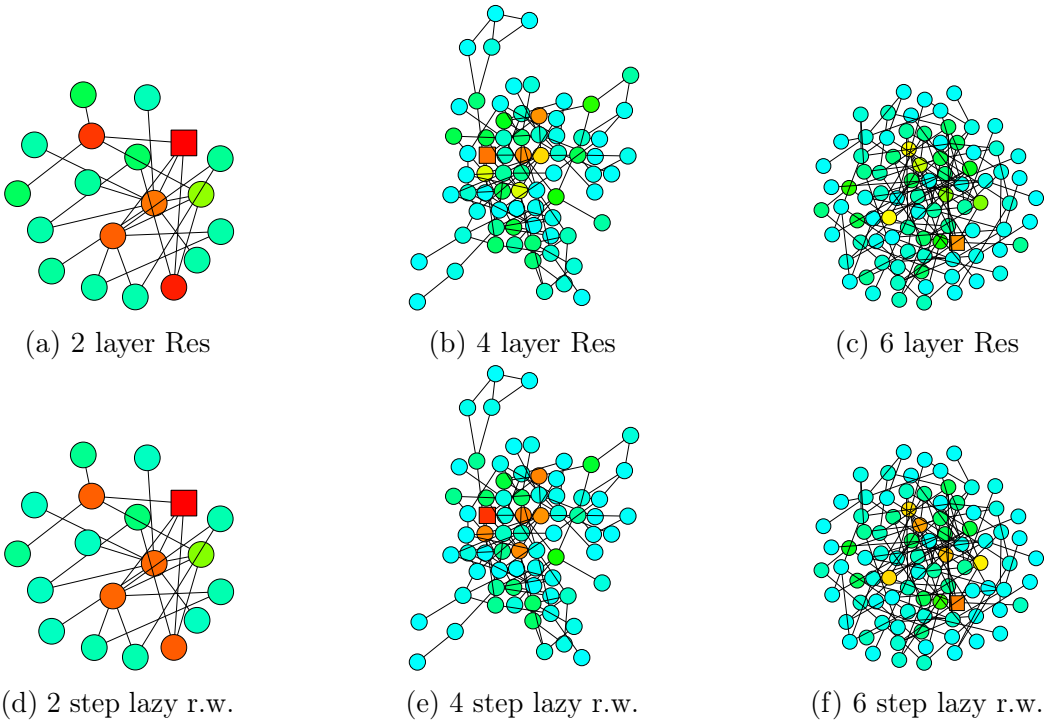


(a) 2 layer GCN (b) 4 layer GCN (c) 6 layer GCN

(d) 2 step r.w. (e) 4 step r.w. (f) 6 step r.w.

Figure 2-1: Influence distributions of GCNs and random walk distributions starting at the square node

Empirically, we observe that, despite somewhat simplifying assumptions, our theory is close to what happens in practice. We visualize the heat maps of the influence distributions for a node (labeled square) for trained GCNs, and compare with the random walk distributions starting at the same node. Figure 2-1 shows example results. Darker colors correspond to higher influence probabilities. To show the effect of skip connections, Figure 2-2 visualizes the analogous heat maps for one example—GCN with residual connections. Indeed, we observe that the influence distributions of networks with residual connections approximately correspond to lazy random walks: each step has a higher probability of staying at the current node. Local information is retained with similar probabilities for all nodes in each iteration; this cannot adapt to diverse needs of specific upper-layer nodes. Further visualizations may be found in the appendix.



(a) 2 layer Res (b) 4 layer Res (c) 6 layer Res

(d) 2 step lazy r.w. (e) 4 step lazy r.w. (f) 6 step lazy r.w.

Figure 2-2: Influence distributions of GCNs with residual connections and random walk distributions with lazy factor 0.4

2.1.2 Fast Collapse on Expanders

To better understand the implication of Theorem 1 and the limitations of the corresponding neighborhood aggregation algorithms, we revisit the scenario of learning on a social network shown in Figure 1-1. Random walks starting inside an expander converge rapidly in $O(\log |V|)$ steps to an almost-uniform distribution [16]. After $O(\log |V|)$ iterations of neighborhood aggregation, by Theorem 1 the representation of every node is influenced almost equally by any other node in the expander. Thus, the node representations will be representative of the global graph and carry limited information about individual nodes. In contrast, random walks starting at the bounded tree-width (almost-tree) part converge slowly, i.e., the features retain more local information. Models that impose a fixed random walk distribution inherit these discrepancies in the speed of expansion and influence neighborhoods, which may not lead to the best representations for all nodes.

2.2 Answers to the Puzzles

Building upon our analysis of influence distributions and random walks, we can now answer the first two questions raised in our introduction.

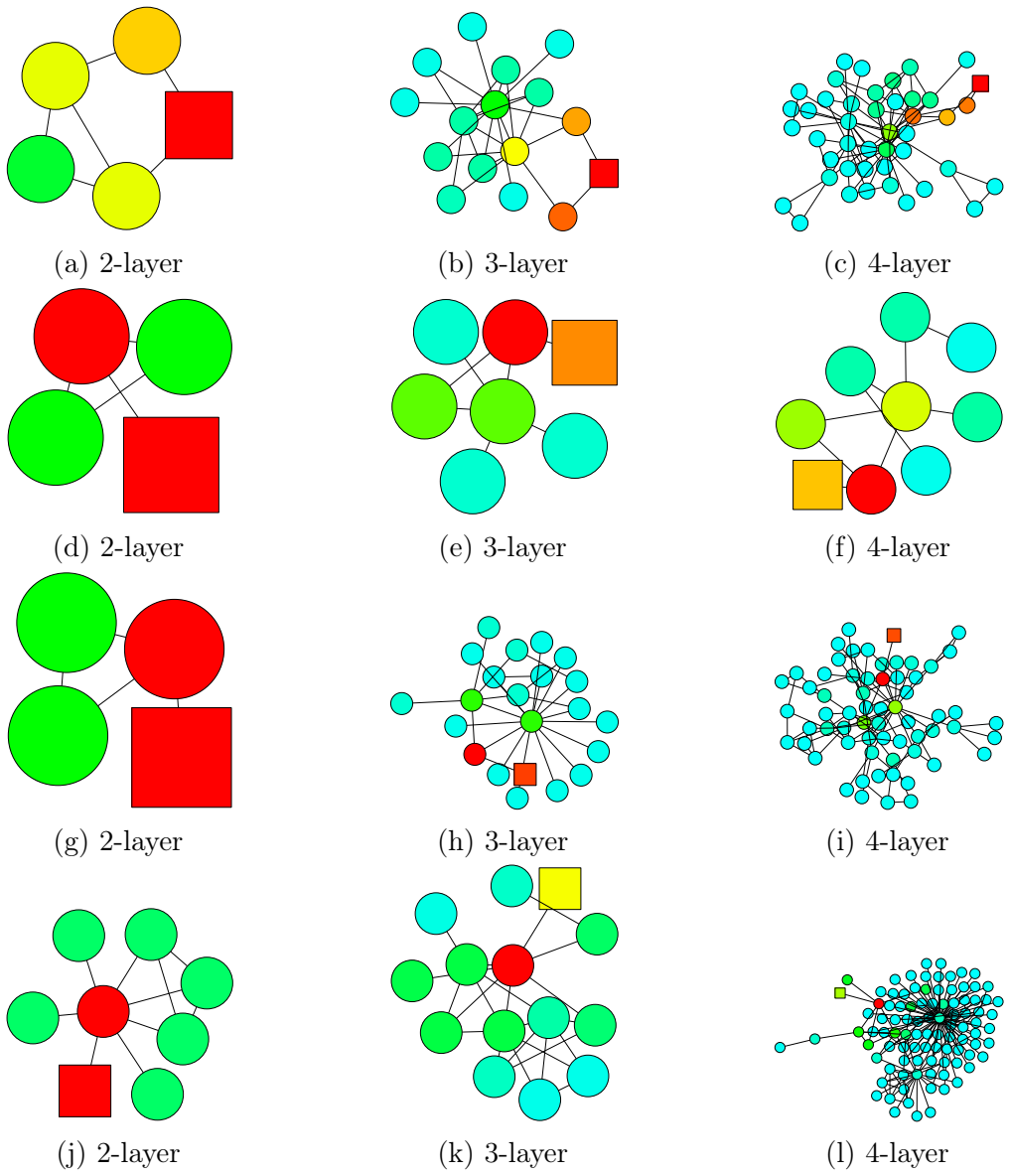
First, what leads to the "curse of depth" in GNNs? The answer, as we have already seen in the analysis of fast collapse on expanders, is that more layers will lead to rapid expansion of influence distribution on expanders and thus possible degradation in performance. Though more layers may lead to better performance in bounded treewidth parts, when a fixed number of layers ia applied and thus no structure-aware adaptivity is available, the averaged best performance may be achieved with only two GNN layers.

Second, why does residual connection not resolve the problem for learning graphs as in computer vision? As we have seen in Figure 2-2, skip connections like residual connections play a role of "laziness" in the random walk expansion. The laziness, however, cannot fundamentally resolve, but at most alleviate, the problem of discrepancy in expansion speed in different graph structures. Therefore, a new architecture to address the problem is desirable. Moreover, in computer vision, the images (formulated as regular grid graphs) are far from expanders and the structures are the same almost everywhere. Thus, the same problem does not occur in computer vision.

Next, we empirically validate our theory and answers via visualization of misclassfication in node classification tasks. We demonstrate subgraph structures where GCN models with 2 layers tend to make misclassification, whereas models with 3 or 4 layers are able to make the correct prediction and vice versa, with real dataset. These visualization results further complement and support the theory illustrated in Figure 1-1 and Theorem 1. As we see in Figure 2-3, a model with pre-fixed effective range priors, which looks at 2-hop neighbors, tends to make incorrect prediction if the local subgraph structure is tree-like (bounded treewidth). Thus, it would be desirable to look beyond the direct neighbors and draw information from nodes that are 3 or 4 hops away so as to learn a better representation of the local community. On the other hand, as we see in Figure 2-4, a model with pre-xied effective range priors, which looks

at 3 or 4-hop neighbors, may happen to draw much information from less relevant neighbors and thus cannot learn the right representations, which are necessary for the correct prediction. In the subgraph structures where the random walk expansion explodes rapidly, models with 3 or 4 prefixed layers are essentially taking into account every node. Such global representations might not be ideal for the prediction for the node. In another scenario, despite possessing the locally bounded treewidth structure, because of the "bridge-like" structures, looking at distant nodes might imply drawing information from a completely different community, which would act like noises and influence the prediction results.

To answer the third question "Are models with two layers really the best, or can we do better?", in the next chapter, we develop an architecture Jumping Knowledge Networks (JK-Nets) that overcomes the problem of existing GNNs by adaptively adapting to the subgraph structures.



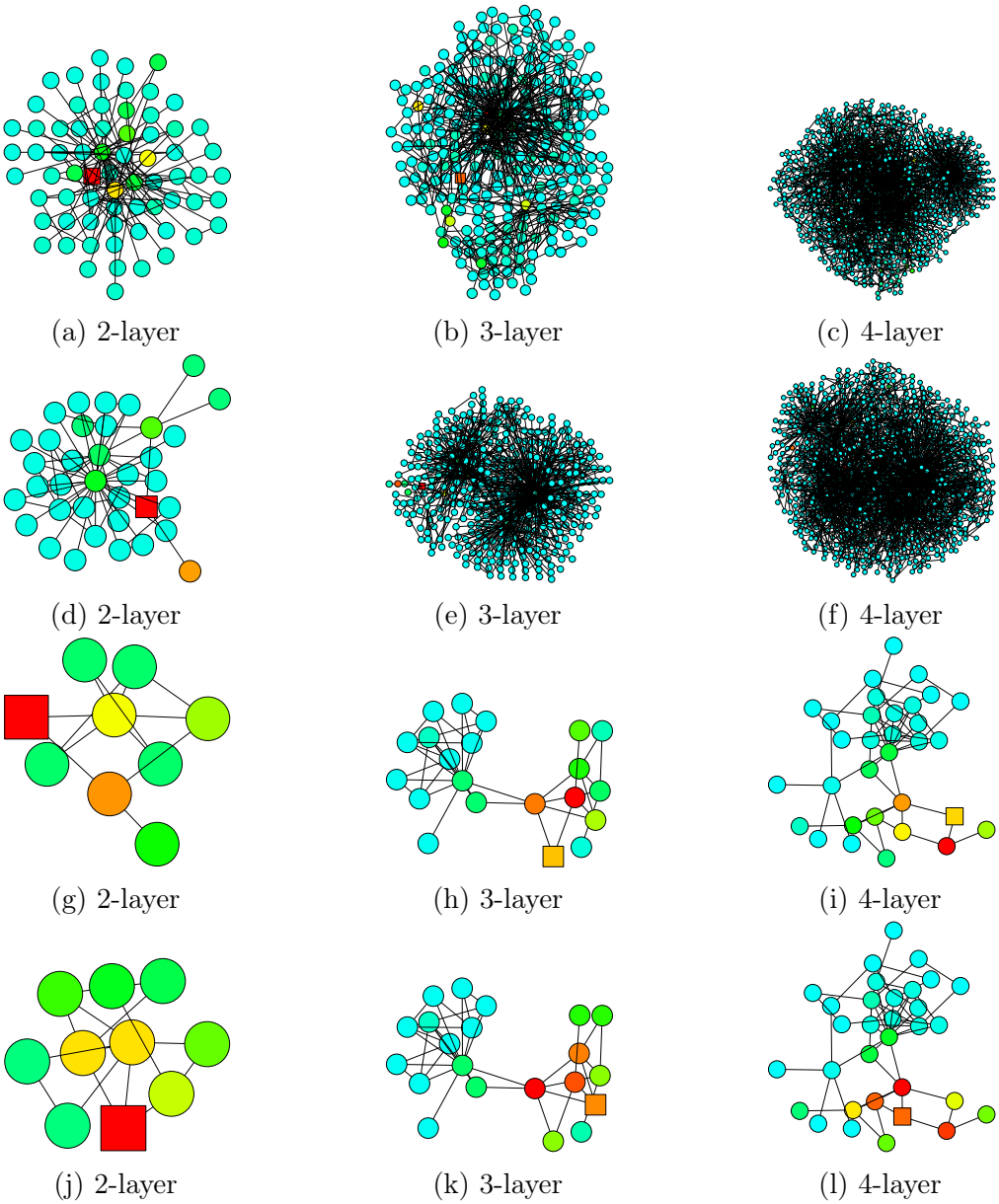
(a) 2-layer (b) 3-layer (c) 4-layer

(d) 2-layer (e) 3-layer (f) 4-layer

(g) 2-layer (h) 3-layer (i) 4-layer

(j) 2-layer (k) 3-layer (l) 4-layer

Figure 2-3: Subgraph structures where 2-layer GCNs make misclassification, whereas 3 and 4-layer GCNs make the correct prediction.



(a) 2-layer (b) 3-layer (c) 4-layer

(d) 2-layer (e) 3-layer (f) 4-layer

(g) 2-layer (h) 3-layer (i) 4-layer

(j) 2-layer (k) 3-layer (l) 4-layer

Figure 2-4: Subgraph structures where 3, 4-layer GCNs make misclassification, whereas 2-layer GCNs make the correct prediction.

Chapter 3

Models

In this chapter, we develop an architecture Jumping Knowledge Networks (JK-Nets), which can be applied on top of any general Graph Neural Network models and enable adaptively learning the node representations and appropriate influence radii with respect to the subgraph structures. JK-Nets will resolve the problem of "curse of depth", and enable learning graphs with *deep* GNNs.

3.1 Jumping Knowledge Networks

Our theory of graph structures, random walks and GNNs raise the question whether the fixed but structure-dependent influence radius size induced by common aggregation schemes really achieves the best representations for all nodes and tasks. Large radii may lead to too much averaging, while small radii may lead to instabilities or insufficient information aggregation. Hence, we propose two simple yet powerful architectural changes – jump connections and a subsequent selective but adaptive aggregation mechanism.

Figure 3-1 illustrates the main idea: as in common neighborhood aggregation networks, each layer increases the size of the influence distribution by aggregating neighborhoods from the previous layer. At the last layer, for each node, we carefully select from all of those itermediate representations (which "jump" to the last layer), potentially combining a few. If this is done independently for each node, then the

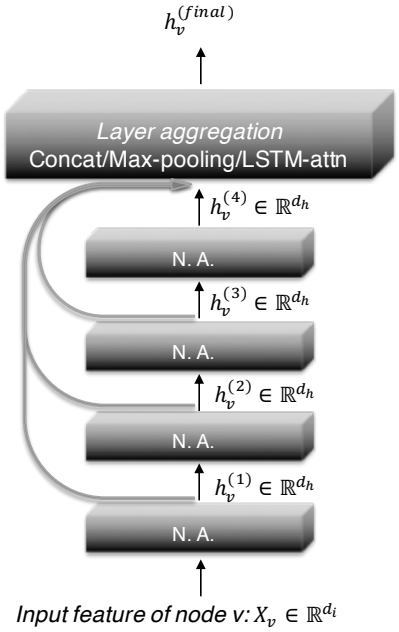


Figure 3-1: A 4-layer Jumping Knowledge Network (JK-Net). N.A. stands for neighborhood aggregation.

model can adapt the effective neighborhood size for each node as needed, resulting in exactly the desired adaptivity.

Our model permits general layer-aggregation mechanisms. We explore three approaches; others are possible too. Let $h_v^{(1)}, ..., h_v^{(k)}$ be the jumping representations of node v (from k layers) that are to be aggregated.

Concatenation. A concatenation $\left[h_v^{(1)}, ..., h_v^{(k)}\right]$ is the most straightforward way to combine the layers, after which we may perform a linear transformation. If the transformation weights are shared across graph nodes, this approach is not node-adaptive. Instead, it optimizes the weights to combine the subgraph features in a way that works best for the dataset overall. One may expect concatenation to be suitable for small graphs and graphs with regular structure that require less adaptivity; also because weight-sharing helps reduce overfitting.

Max-pooling. An element-wise max $\left(h_v^{(1)}, ..., h_v^{(k)}\right)$ selects the most informative layer *for each feature coordinate*. For example, feature coordinates that represent more local properties can use the feature coordinates learned from the close neighbors and those representing global status would favor features from the higher-up layers. Max-

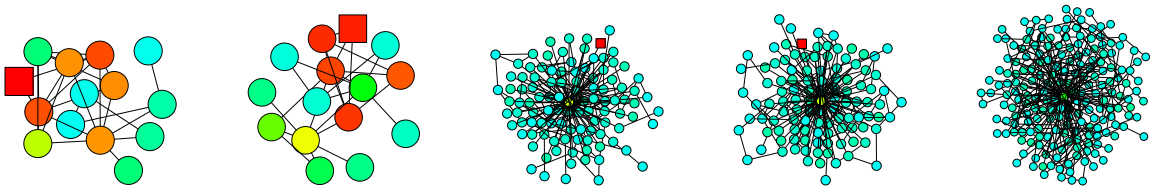
pooling is adaptive and has the advantage that it does not introduce any additional parameters to learn.

LSTM-attention. An attention mechanism identifies the most useful neighborhood ranges for each node v by computing an attention score $s_v^{(l)}$ for each layer l $\left(\sum_l s_v^{(l)} = 1\right)$, which represents the importance of the feature learned on the l-th layer for node v. The aggregated representation for node v is a weighted average of the layer features $\sum_l s_v^{(l)} \cdot h_v^{(l)}$. For LSTM attention, we input $h_v^{(1)}, ..., h_v^{(k)}$ into a bi-directional LSTM [15] and generate the forward-LSTM and backward-LSTM hidden features $f_v^{(l)}$ and $b_v^{(l)}$ for each layer l. A linear mapping of the concatenated features $[f_v^{(l)}||b_v^{(l)}]$ yields the scalar importance score $s_v^{(l)}$. A Softmax layer applied to $\{s_v^{(l)}\}_{l=1}^k$ yields the attention of node v on its neighborhood in different ranges. Finally we take the sum of $[f_v^{(l)}||b_v^{(l)}]$ weighted by $\texttt{SoftMax}(\{s_v^{(l)}\}_{l=1}^k)$ to get the final layer representation. Another possible implementation combines LSTM with max-pooling. LSTM-attention is node adaptive because the attention scores are different for each node. We shall see that the this approach shines on large complex graphs, although it may overfit on small graphs (fewer training nodes) due to its relatively higher complexity.

3.1.1 JK-Net Learns to Adapt

The key idea for the design of layer-aggregation functions is to determine the importance of a node's subgraph features at different ranges after looking at the learned features on all layers, rather than to optimize and fix the same weights for all nodes. Under the same assumption on the ReLU activation distribution as in Theorem 1, we show below that layer-wise max-pooling implicitly learns the influence locality adaptively for different nodes. The proof for layer-wise attention follows similarly.

Proposition 1. *Assume that paths of the same length in the computation graph are activated with the same probability. The influence score $I(x, y)$ for any $x, y \in V$ under a k-layer JK-Net with layer-wise max-pooling is equivalent in expectation to a mixture of $0, .., k$-step random walk distributions on $\widetilde{G}$ at y starting at x, the coefficients of which depend on the values of the layer features $h_x^{(l)}$.*



(a) tree-like (b) tree-like (c) affiliate (d) affiliate (e) hub

Figure 3-2: A 6-layer JK-Net learns to adapt to different subgraph structures

Proof. Let $\left[h_x^{(final)}\right]_i$ be the i-th entry of $h_x^{(final)}$, the feature after layer aggregation. For any node y, we have

$$I(x,y) = \sum_i \left\| \frac{\partial \left[h_x^{(final)}\right]_i}{\partial h_y^{(0)}} \right\|_1 = \sum_i \left\| \frac{\partial \left[h_x^{(j_i)}\right]_i}{\partial h_y^{(0)}} \right\|_1$$

where $j_i = \underset{l}{\text{argmax}} \left(\left[h_x^{(l)}\right]_i \right)$. By Theorem 1, we have

$$\mathbb{E}\left[I(x,y)\right] = \sum_l c_x^l \cdot z_l \cdot \mathbb{E}\left[I_x(y)^{(l)}\right]$$

where $I_x(y)$ is the l-step random walk probability at y, z_l is a normalization factor and c_x^l is the fraction of entries of $h_x^{(l)}$ being chosen by max-pooling. By Theorem 1, $\mathbb{E}\left[I_x(y)^{(l)}\right]$ is equivalent to the l-step random walk probability at y starting at x. □

Contrasting this result with the influence distributions of other aggregation mechanisms, we see that JK-networks indeed differ in their node-wise adaptivity of neighborhood ranges.

Figure 3-2 illustrates how a 6-layer JK-Net with max-pooling aggregation learns to adapt to different subgraph structures on a citation network. Within a tree-like structure, the influence stays in the "small community" the node belongs to. In contrast, 6-layer models whose influence distributions follow random walks, e.g. GCNs, would reach out too far into irrelevant parts of the graph, and models with few layers

may not be able to cover the entire "community", as illustrated in Figure 1-1, and Figures 2-3, 2-4. For a node affiliated to a "hub", which presumably plays the role of connecting different types of nodes, JK-Net learns to put most influence on the node itself and otherwise spreads out the influence. GCNs, however, would not capture the importance of the node's own features in such a structure because the probability at an affiliate node is small after a few random walk steps. For hubs, JK-Net spreads out the influence across the neighboring nodes in a reasonable range, which makes sense because the nodes connected to the hubs are presumably as informative as the hubs' own features. For comparison, Table A.1 in the appendix includes more visualizations of how models with random walk priors behave.

3.1.2 Intermediate Layer Aggregation and Structures

Looking at Figure 3-1, one may wonder whether the same inter-layer connections could be drawn between all layers. The resulting architecture is approximately a graph correspondent of DenseNets, which were introduced for computer vision problems [18], if the layer-wise concatenation aggregation is applied. This version, however, would require many more features to learn. Viewing the DenseNet setting (images) from a graph-theoretic perspective, images correspond to regular, in fact, near-planar graphs. Such graphs are far from being expanders, and do not pose the challenges of graphs with varying subgraph structures. Indeed, as we shall see, models with concatenation aggregation, e.g., JK-Concat, perform well on graphs with more regular structures such as images and well-structured communities. On the other hand, on graphs with more diverse structures, the node-wise adaptive methods like JK-LSTM perform better. As a more general framework, JK-Net admits general layer-wise aggregation models and enables better structure-aware representations on graphs with complex structures.

3.2 Other Related Work

Traditional node embedding methods often rely on subgraph summary statistics. In DeepWalk [31] and node2vec [10], nodes have similar embeddings if they tend to

co-occur on short random walks. LINE [37] preserves the first and second order proximity between nodes, on top of which LOG [28] also preserves node PageRank. Our model distinguishes from these methods via learning node representations in an end-to-end task-specific fashion.

Spectral graph convolutional neural networks apply convolution on graphs by using the graph Laplacian eigenvectors as the Fourier atoms [4, 36, 7]. A major drawback of the spectral methods, compared to spatial approaches like neighborhood-aggregation, is that the graph Laplacian needs to be known in advance. Hence, they cannot generalize to unseen graphs. Note that despite being derived from an approximation of the spectral filters, GCN is spatial in nature. Diffussion CNNs [2] produce multi-scale features by multiplying the node features with powers of random walk matrices. JK-Net is more flexible, as it allows generic aggregations of different neighborhood representations and, as opposed to random walks (or the linear aggregations of DCNNs), uses nonlinearities in each layer that make it more expressive.

Chapter 4

Experiments

4.1 Empirical Results

We evaluate JK-Nets on four node classification benchmark datasets. (I) The task on citation networks (Citeseer, Cora) [34] is to classify academic papers into different subjects. The dataset contains bag-of-words features for each document (node) and citation links (edges) between documents. (II) On Reddit [11], the task is to predict the community to which different Reddit posts belong. Reddit is an online discussion forum where users comment in different topical communities. Two posts (nodes) are connected if some user commented on both posts. The dataset contains word vectors as node features. (III) For protein-protein interaction networks (PPI) [11], the task is to classify protein functions. PPI consists of 24 graphs, each corresponds to a human tissue. Each node has positional gene sets, motif gene sets and immunological signatures as features and gene ontology sets as labels. 20 graphs are used for training, 2 graphs are used for validation and the rest for testing. Statistics of the datasets are summarized in Table 4.1.

Settings. In the *transductive setting*, we are only allowed to access a subset of nodes in one graph as training data, and validate/test on others. Our experiments on Citeseer, Cora and Reddit are transductive. In the *inductive setting*, we use a number of full graphs as training data and use other completely unseen graphs as validation/testing data. Our experiments on PPI are inductive.

We compare against three baselines: Graph Convolutional Networks (GCN) [22], GraphSAGE [11] and Graph Attention Networks (GAT) [39].

Dataset	Nodes	Edges	Classes	Features
Citeseer	3,327	4,732	6	3,703
Cora	2,708	5,429	7	1,433
Reddit	232,965	avg deg 492	50	300
PPI	56,944	818,716	121	50

Table 4.1: Dataset statistics

4.1.1 Citeseer & Cora

For experiments on Citeseer and Cora, we choose GCN as the base model since on our data split, it is outperforming GAT. We construct JK-Nets by choosing MaxPooling (JK-MaxPool), Concatenation (JK-Concat), or LSTM-attention (JK-LSTM) as final aggregation layer. When taking the final aggregation, besides normal graph convolutional layers, we also take the first linear-transformed representation into account. The final prediction is done via a fully connected layer on top of the final aggregated representation. We split nodes in each graph into 60%, 20% and 20% for training, validation and testing. We vary the number of layers from 1 to 6 for each model and choose the best performing model with respect to the validation set. Throughout the experiments, we use the Adam optimizer [21] with learning rate 0.005. We fix the dropout rate to be 0.5, the dimension of hidden features to be within $\{16, 32\}$, and add an $L2$ regularization of 0.0005 on model parameters. The results are shown in Table 4.2.

Results. We observe in Table 4.2 that JK-Nets outperform both GCN and GAT baselines in terms of prediction accuracy. Though JK-Nets perform well in general, there is no consistent winner and performance varies slightly across datasets.

Taking a closer look at results on Cora, both GCN and GAT achieve their best accuracies with only 2 or 3 layers, suggesting that local information is a stronger signal for classification than global ones. However, the fact that JK-Nets achieve the

Model	Citeseer	Model	Cora
GCN (2)	77.3 (1.3)	GCN (2)	88.2 (0.7)
GAT (2)	76.2 (0.8)	GAT (3)	87.7 (0.3)
JK-MaxPool (1)	77.7 (0.5)	JK-Maxpool (6)	**89.6** (0.5)
JK-Concat (1)	**78.3** (0.8)	JK-Concat (6)	89.1 (1.1)
JK-LSTM (2)	74.7 (0.9)	JK-LSTM (1)	85.8 (1.0)

Table 4.2: Results of GCN-based JK-Nets on Citeseer and Cora. The baselines are GCN and GAT. The number in parentheses next to the model name indicates the best-performing number of layers among 1 to 6. Accuracy and standard deviation are computed from 3 random data splits.

best performance with 6 layers indicates that global together with local information will help boost performance. This is where models like JK-Nets can be particularly beneficial. LSTM-attention may not be suitable for such small graphs because of its relatively high complexity.

4.1.2 Reddit

The Reddit data is too large to be handled well by current implementations of GCN or GAT. Hence, we use the more scalable GraphSAGE as the base model for JK-Net. It has skip connections and different modes of node aggregation. We experiment with Mean and MaxPool node aggregators, which take mean and max-pooling of a *linear transformation* of representations of the sampled neighbors. Combining each of GraphSAGE modes with MaxPooling, Concatenation or LSTM-attention as the last aggregation layer gives 6 JK-Net variants. We follow exactly the same setting of GraphSAGE as in the original paper [11], where the model consists of 2 hidden layers, each with 128 hidden units and is trained with Adam with learning rate of 0.01 and no weight decay. Results are shown in Table 4.3.

Results. With MaxPool as node aggregator and Concat as layer aggregator, JK-Net achieves the best Micro-F1 score among GarphSAGE and JK-Net variants. Note that the original GraphSAGE already performs fairly well with a Micro-F1 of 0.95. JK-Net reduces the error by 30%. The communities in the Reddit dataset were

Node \ JK	GraphSAGE	Maxpool	Concat	LSTM
Mean	0.950	0.953	0.955	0.950
MaxPool	0.948	0.924	**0.965**	0.877

Table 4.3: Results of GraphSAGE-based JK-Nets on Reddit. The baseline is GraphSAGE. Model performance is measured in Micro-F1 score. Each column shows the results of a JK-Net variant. For all models, the number of layers is fixed to 2.

explicitly chosen from the well-behaved middle-sized communities to avoid the noisy cores and tree-like small communities [11]. As a result, this graph is more regular than the original Reddit data, and hence not exhibit the problems of varying subgraph structures. In such a case, the added flexibility of the node-specific neighborhood choices may not be as relevant, and the stabilizing properties of concatenation instead come into play.

4.1.3 Protein-to-protein Interaction (PPI)

We demonstrate the power of adaptive JK-Nets, e.g., JK-LSTM, with experiments on the PPI data, where the subgraphs have more diverse and complex structures than those in the Reddit community detection dataset. We use both GraphSAGE and GAT as base models for JK-Net. The implementation of GraphSAGE and GAT are quite different: GraphSAGE is sample-based, where neighbors of a node are sampled to be a fixed number, while GAT considers all neighbors. Such differences cause large gaps in terms of both scalability and performances. Given that GraphSAGE scales to much larger graphs, it appears particularly valuable to evaluate how much JK-Net can improve upon GraphSAGE.

For GraphSAGE we follow the setup as in the Reddit experiments, except that we use 3 layers when possible, and compare the performance after 10 and 30 epochs of training. The results are shown in Table 4.4. For GAT and its JK-Net variants we stack two hidden layers with 4 attention heads computing 256 features (for a total of 1024 features), and a final prediction layer with 6 attention heads computing 121 features each. They are further averaged and input into sigmoid activations. We

Node \ JK	SAGE	MaxPool	Concat	LSTM
Mean (10 epochs)	0.644	0.658	0.667	**0.721**
Mean (30 epochs)	0.690	0.713	0.694	**0.818**
MaxPool (10 epochs)	0.668	0.671	0.687	0.621*

Table 4.4: Results of GraphSAGE-based JK-Net on the PPI data. The baseline is GraphSAGE (SAGE). Each column, excluding SAGE, represents a JK-Net with different layer aggregation. All models use 3 layers, except for those with "*", whose number of layers is set to 2 due to GPU memory constraints. 0.6 is the corresponding 2-layer GraphSAGE performance.

Model	PPI
MLP	0.422
GAT	0.968 (0.002)
JK-Concat (2)	0.959 (0.003)
JK-LSTM (3)	0.969 (0.006)
JK-Dense-Concat (2)*	0.956 (0.004)
JK-Dense-LSTM (2)*	**0.976** (0.007)

Table 4.5: Micro-F1 scores of GAT-based JK-Nets on the PPI data. The baselines are GAT and MLP (Multilayer Perceptron). While the number of layers for JK-Concat and JK-LSTM are chosen from $\{2, 3\}$, the ones for JK-Dense-Concat and JK-Dense-LSTM are directly set to 2 due to GPU memory constraints.

employ skip connections across intermediate attentional layers. These models are trained with Batch-size 2 and Adam optimizer with learning rate of 0.005. The results are shown in Table 4.5.

Results. JK-Nets with the LSTM-attention aggregators outperform the non-adaptive models GraphSAGE, GAT and JK-Nets with concatenation aggregators. In particular, JK-LSTM outperforms GraphSAGE by 0.128 in terms of micro-F1 score after 30 epochs of training. Structure-aware node adaptive models are especially beneficial on such complex graphs with diverse structures.

Chapter 5

Conclusion

In this thesis, motivated by observations that reveal great differences in neighborhood information ranges for graph node embeddings in different graph structures, we propose a new aggregation scheme – Jumping Knowledge Networks (JK-Nets) for Graph Neural Networks (GNNs) that can adapt neigborhood ranges to nodes individually by exploiting the subgraph structures. This JK-network can improve representations in particular for graphs that have subgraphs of diverse local structure, and may hence not be well captured by fixed numbers of neighborhood aggregations. We also present an analysis framework for the influence distributions of GNNs, which relate the influence distributions of learned node representations to random walks and their neighborhood subgraph structures. Our model analysis provides insight into when a model would fail and how our model can help in these cases.

Interesting directions for future work include exploring other layer aggregators and studying the effect of the combination of various layer-wise and node-wise aggregators on different types of graph structures. It would also be interesting to see empirically how JK-Nets improve the training and representation learning of entire graphs for tasks like graph classification.

Appendix A

Visualization Results

We describe the details of the heat maps and present more visualization results. The colors of the nodes in the heat maps correspond to their probability masses of either the influence distribution or random walk distribution as shown in Figure A-1. As we see, the shallower the color is, the smaller the probability mass. We use the same color for probabilities over 0.2 for better visual effects because there are few nodes with influence probability masses over 0.2. Nodes with probability mass less than 0.001 are not shown in the heat maps.

In Table A.1 and Tabel A.2, we present more visualization results to compare the 1) influence distributions under GCNs and the random walk distributions, 2) influence distributions under GCNs with residual connections and lazy random walk distributions. The visualization results further empirically validate the equivalence of random walks and GCN influence distributions in Theorem 1. The nodes being influenced and the random walk starting node are labeled square. The influence distributions for the nodes in Figure A.1 are computed according to Definition 2.1.1, under the same trained GCN (Res) models with $2, 4, 6$ layers respectively. We use the hyper-parameters as described in [22] for training the models. The graph (dataset) is taken from the Cora citation network as described in section 4.1. We compute the random walk distributions according to Definition 2.1.2 on the graph $\widetilde{G}$. The lazy random walks all share the same lazy factor 0.4, i.e. there's an extra 0.4 probability of staying at the current node at each step. This probability is chosen for visual

Figure A-1: Color and probability correpondency for the heat maps

comparison with the GCN-ResNet. Note that the GCN and random walk colors may differ for nodes that have particularly large degrees because the models we are running follow Equation 1.3, which assigns less weight to nodes that have larger degrees, rather than Equation 1.4. The visualization in Figure 3-2 has the same setting as mentioned above. It is trained for the Cora dataset with a 6-layer JK Net with maxpooling layer aggregation.

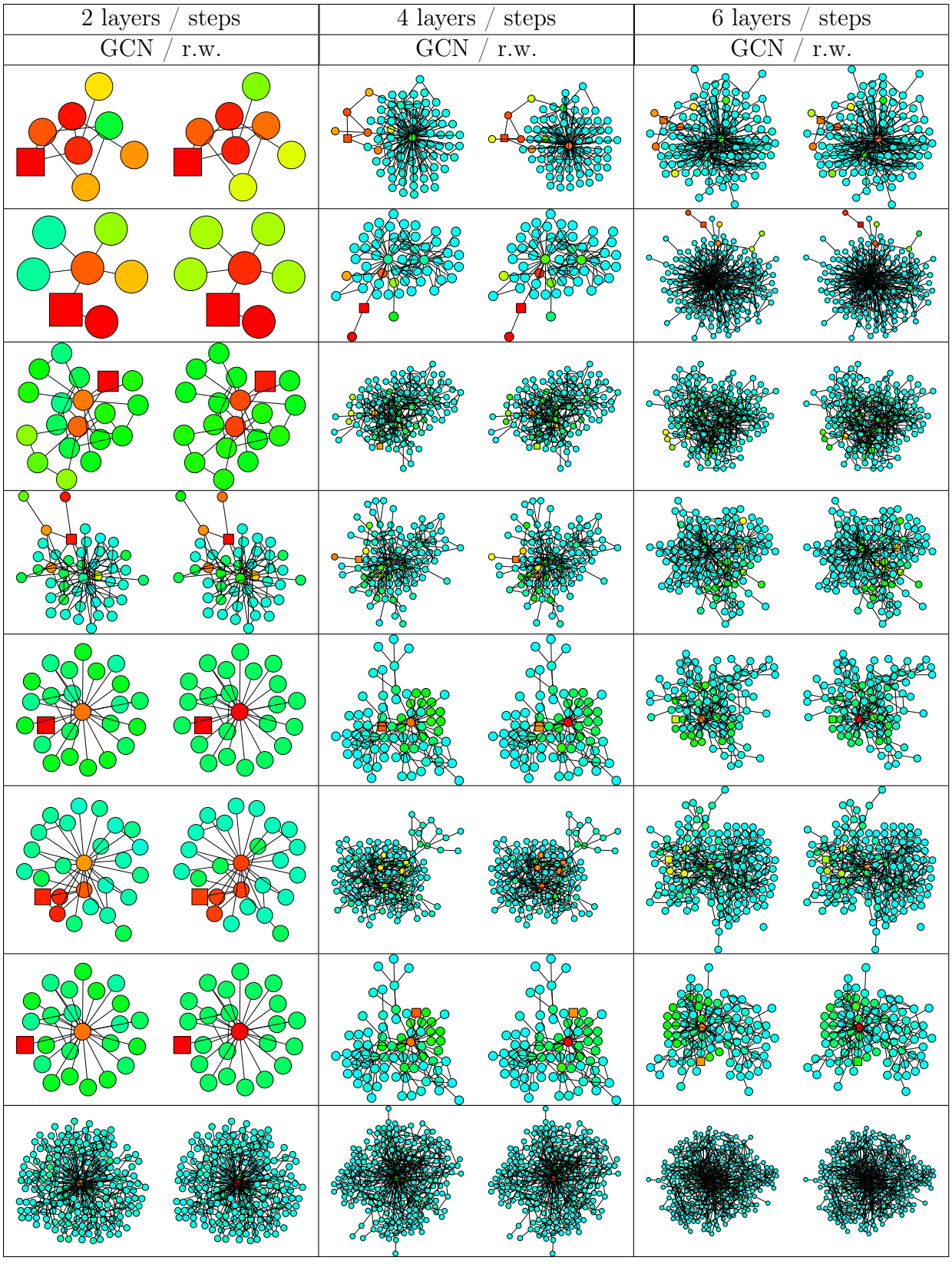


Table A.1: Influence distributions for (more) nodes under GCN and random walk distributions

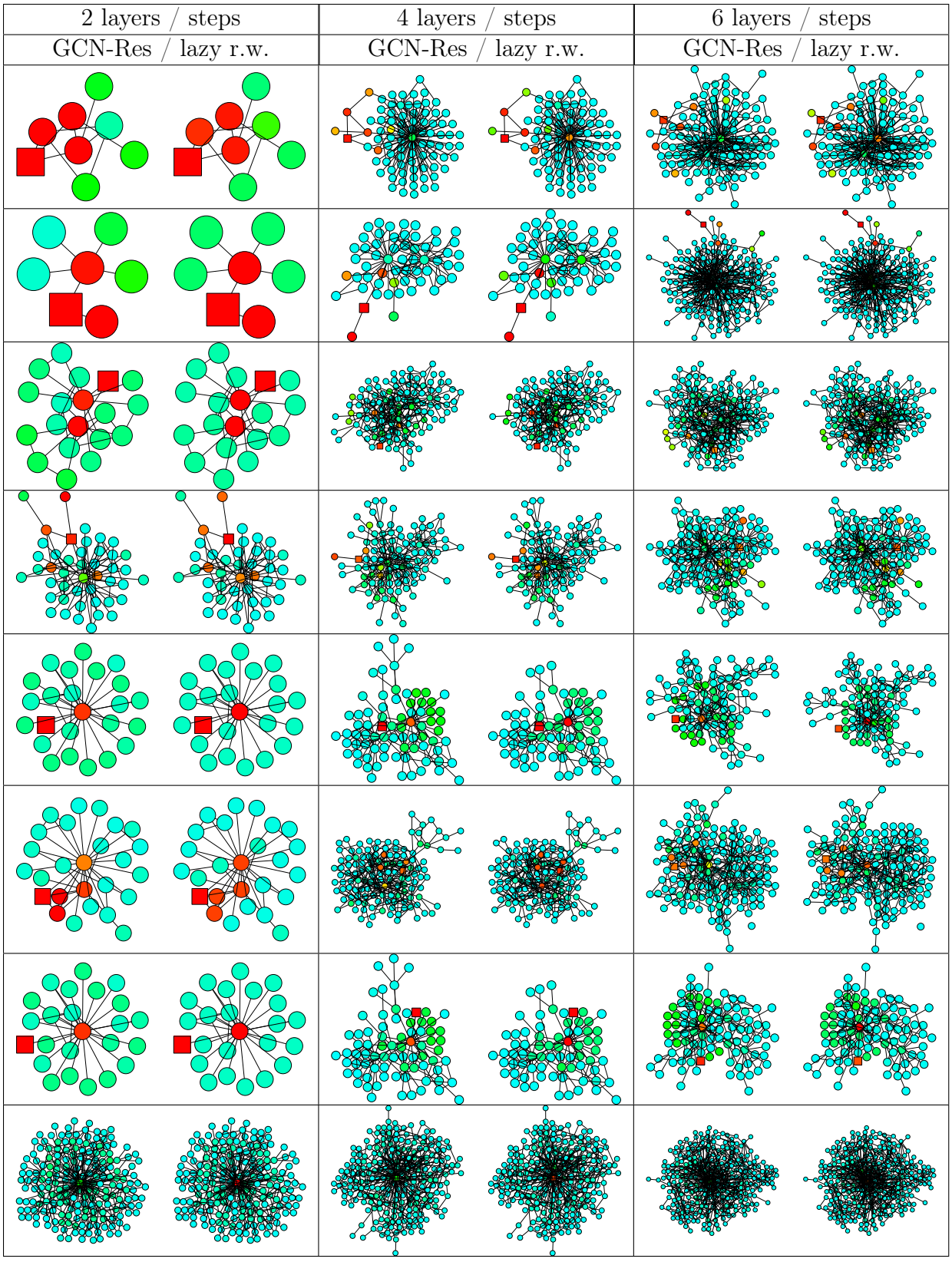


Table A.2: Influence distributions for (more) nodes under GCN with residual connections (GCN-Res) and lazy random walk distributions

Bibliography

[1] Stefan Arnborg, Derek G Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in ak-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

[2] James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1993–2001, 2016.

[3] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in Neural Information Processing Systems (NIPS)*, pages 4502–4510, 2016.

[4] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations (ICLR)*, 2014.

[5] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. In *Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, 2014.

[6] Anna Choromanska, Yann LeCun, and Gérard Ben Arous. Open problem: The landscape of the loss surfaces of multilayer networks. In *Conference on Learning Theory (COLT)*, pages 1756–1760, 2015.

[7] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3844–3852, 2016.

[8] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. pages 2224–2232, 2015.

[9] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning (ICML)*, pages 1273–1272, 2017.

[10] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pages 855–864, 2016.

[11] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1025–1035, 2017.

[12] David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.

[13] 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.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645, 2016.

[15] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[16] Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43(4):439–561, 2006.

[17] Yedid Hoshen. Vain: Attentional multi-agent predictive modeling. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2698–2708, 2017.

[18] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017.

[19] Kenji Kawaguchi. Deep learning without poor local minima. In *Advances in Neural Information Processing Systems (NIPS)*, pages 586–594, 2016.

[20] Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. Molecular graph convolutions: moving beyond fingerprints. *Journal of computer-aided molecular design*, 30(8):595–608, 2016.

[21] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[22] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[23] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning (ICML)*, pages 1885–1894, 2017.

[24] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

[25] Jure Leskovec and Julian J Mcauley. Learning to discover social circles in ego networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 539–547, 2012.

[26] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. In *International Conference on Learning Representations (ICLR)*, 2016.

[27] László Lovász. Random walks on graphs. *Combinatorics, Paul erdos is eighty*, 2:1–46, 1993.

[28] Yao Ma, Suhang Wang, Zhaochun Ren, Dawei Yin, and Jiliang Tang. Preserving local and global information for network embedding. *arXiv preprint arXiv:1710.07266*, 2017.

[29] Takanori Maehara, Takuya Akiba, Yoichi Iwata, and Ken-ichi Kawarabayashi. Computing personalized pagerank quickly by exploiting graph structures. *Proceedings of the VLDB Endowment*, 7(12):1023–1034, 2014.

[30] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5425–5434, 2017.

[31] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pages 701–710, 2014.

[32] Trang Pham, Truyen Tran, Dinh Q Phung, and Svetha Venkatesh. Column networks for collective classification. In *AAAI Conference on Artificial Intelligence*, pages 2485–2491, 2017.

[33] 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.

[34] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93, 2008.

[35] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561, 2011.

[36] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98, 2013.

[37] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the International World Wide Web Conference (WWW)*, pages 1067–1077, 2015.

[38] Anastasios A Tsonis, Kyle L Swanson, and Paul J Roebber. What do networks have to do with climate? *Bulletin of the American Meteorological Society*, 87(5):585–595, 2006.

[39] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations (ICLR)*, 2018.

[40] Saurabh Verma and Zhi-Li Zhang. Graph capsule convolutional neural networks. *arXiv preprint arXiv:1805.08090*, 2018.

[41] Boris Weisfeiler and AA Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 2(9):12–16, 1968.

[42] Pinar Yanardag and SVN Vishwanathan. A structural smoothing framework for robust graph comparison. In *Advances in Neural Information Processing Systems*, pages 2134–2142, 2015.

[43] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems (NIPS)*, 2018.

[44] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *AAAI Conference on Artificial Intelligence*, pages 4438–4445, 2018.