NEURAL REPRESENTATION LEARNING FOR SEMI-SUPERVISED NODE CLASSIFICATION AND EXPLAINABILITY

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Hogun Park

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2020

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

To my beloved family.

ACKNOWLEDGMENTS

This dissertation could not be realized without the help of many people. First of all, I would like to express my sincere gratitude to my advisor, Jennifer Neville. She brought me into the wonderful world of research. She was also a great mentor who gave timely advice during my Ph.D. study. I could learn about the true meaning of passion and dedication from her. I was also able to get valuable advice on my dissertation from my committee members, Chris Clifton, Dan Goldwasser, and Ming Yin. I really appreciate their insightful questions and thoughtful comments. I enjoyed working with them and benefited enormously from the interactions with them.

I thank my lab mates, who make my graduate life a very memorable experience: Nesreen Ahmed, Iman Alodah, Sait Celebi, Timothy Fond, Mahak Goindani, Guilherme Gomes, Pablo Robles-Granda, Mengyue Hang, Suvidha Kancharla, Ellen Lai, Ying-Chun Lin, Jason Meng, John Moore, Sebastian Moreno, Joel Pfeiffer, Xi Tan, Jiasen Yang, Giselle Zeno, and Shandian Zhe.

Many of my friends in PKCS also helped me a lot to keep motivated. The memory of working out together remains a precious memory. I would also like to thank Jaemun Byun, Hyungu Jeong, Eunjoo Kang, Jooho Kim, Tony Kim, Danny Oh, and Frank Suarez-Roman who encouraged and supported me a lot. I also thank my good friend, Koray Mancuhan, for being supportive and giving advice when I have a hard time.

Lastly, I would like to thank my family. Without their dedicated sacrifice and support, I couldn't have come this far. In particular, I am deeply grateful to my wife, Hye Young Sohn for her love, understanding, and encouragement. Without her sacrifice and love, nothing was possible. I would also like to thank my parents, younger sister, sister-in-law, brother-in-law, and my two sons–Jinhun and Jinwoo. Moreover, my mother-in-law and father-in-law stayed in the U.S. for a long time to help us. Without them, it would have been a big challenge.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

ABSTRACT

Park, Hogun Ph.D., Purdue University, August 2020. Neural Representation Learning for Semi-Supervised Node Classification and Explainability. Major Professor: Jennifer Neville.

Many real-world domains are relational, consisting of objects (e.g., users and papers) linked to each other in various ways. Because class labels in graphs are often only available for a subset of the nodes, semi-supervised learning for graphs has been studied extensively to predict the unobserved class labels. For example, we can predict political views in a partially labeled social graph dataset and get expected gross incomes of movies in an actor/movie graph with a few labels. Recently, advances in representation learning for graph data have made great strides for the semi-supervised node classification. However, most of the methods have mainly focused on learning node representations by considering simple relational properties (e.g., random walk) or aggregating nearby attributes, and it is still challenging to learn complex interaction patterns in partially labeled graphs and provide explanations on the learned representations.

In this dissertation, multiple methods are proposed to alleviate both challenges for semi-supervised node classification. First, we propose a graph neural network architecture, REGNN, that leverages local inferences for unlabeled nodes. REGNN performs graph convolution to enable label propagation via high-order paths and predicts class labels for unlabeled nodes. In particular, our proposed attention layer of REGNN measures the *role equivalence* among nodes and effectively reduces the noise, which is generated during the aggregation of observed labels from distant neighbors at various distances. Second, we also propose a neural network architecture that jointly captures both temporal and static *interaction patterns*, which we call Temporal-Static-Graph-Net (TSGNet). The architecture learns a latent representation of each node in order to encode complex interaction patterns. Our key insight is that leveraging both a *static neighbor encoder*, that learns aggregate neighbor patterns, and a *graph neural network-based recurrent unit*, that captures complex interaction patterns, improves the performance of node classification. Lastly, in spite of better performance of representation learning on node classification tasks, neural network-based representation learning models are still less interpretable than the previous relational learning models due to the lack of explanation methods. To address the problem, we show that nodes with high *bridgeness* scores have larger impacts on node embeddings such as DeepWalk [1], LINE [2], Struc2Vec [3], and PTE [4] under perturbation. However, it is computationally heavy to get *bridgeness* scores, and we propose a novel gradient-based explanation method, GRAPH-wGD, to find nodes with high *bridgeness* efficiently. In our evaluations, our proposed architectures (REGNN and TSGNet) for semi-supervised node classification consistently improve predictive performance on real-world datasets. Our GRAPH-wGD also identifies important nodes as global explanations, which significantly change both predicted probabilities on node classification tasks and k-nearest neighbors in the embedding space after perturbing the highly ranked nodes and re-learning low-dimensional node representations for DeepWalk and LINE embedding methods.

1 INTRODUCTION

This dissertation is concerned with learning node representations for attributed or non-attributed graphs and providing explanations of the representations. In a large network such as Facebook social graphs and IMDB actor/movie graphs, our methods find the low-dimensional representation of each user or each movie for semi-supervised node classification and provide explanations that show who contributed the most to the learned representations.

First of all, semi-supervised node classification aims to classify unknown class labels in a partially-labeled graph by leveraging information about both the labeled and unlabeled nodes. The semi-supervised learning exploits the dependencies in the data to jointly make predictions about unlabeled nodes and has been extensively studied in recent years (e.g., [5, 6]) in relational data applications such as social, citation, and biological networks. Recent advances in representation learning for graph data have made great strides for the semi-supervised node classification. Representation learning for semi-supervised node classification is illustrated as in the Figure 1.1 below. We note that an input graph set $G = \{G_1, ..., G_T\}$ is given, which is composed of T number of temporal network snapshots. As in the figure, d-dimensional representation of each node v in G is learned from node representation learning methods (e.g., [1,7]) with (or without) observed class labels Y_L and an attribute matrix X. The representations are leveraged to learn a classifier for node classification. For example, in a Facebook social graph, its task is to predict the political views of unlabeled users.

Prior work on semi-supervised learning in graphs has typically defined relational features via aggregation over the information of neighboring nodes, and then unknown class labels are inferred iteratively using approximate inference algorithms (e.g., Gibbs sampling [8] and composite-likelihood EM [9]). Meanwhile, recent graph neural network models [5, 10, 11] learn relational features using LSTM and CNN-

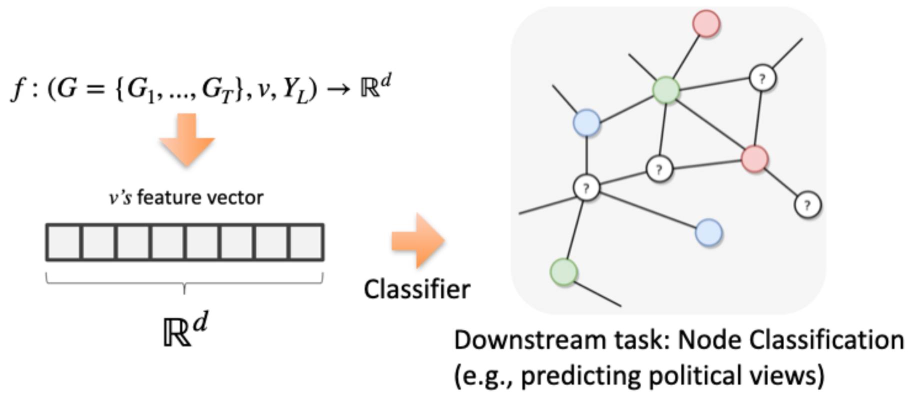

$f : (G = \{G_1, ..., G_T\}, v, Y_L) \rightarrow \mathbb{R}^d$
v's feature vector
$\mathbb{R}^d$
Classifier
?
Downstream task: Node Classification
(e.g., predicting political views)

Figure 1.1.: Illustration of representation learning for node classification tasks. An example of node classification is to predict political views of unlabeled users in a Facebook social graph.

like architectural components. In particular, graph neural networks (GNNs) provide powerful deep neural network architectures, which exploit convolution operators for learning based on nearby neighbors' attributes. For example, the graph convolution operator [5] sums attributes of all the neighbors of nodes with a linear transformation. By stacking multiple layers, it learns a low-dimensional representation of each node. However, recent methods have mainly focused on learning node representations by considering simple relational properties (e.g., random walk) or aggregating information from direct neighbors, and it is still challenging to leverage more complex interaction patterns and provide explanations on the learned representations.

In this dissertation, to move beyond direct neighbors and exploit longer range information in sparse graphs, we consider high-order (or k^{th} order) proximity matrices and extend the existing high-order-based GCNs to leverage inferences about unlabeled nodes via neighborhood at various distances. Since the reasoning with higher-order paths (i.e., larger k) increases the computational complexity of learning. When considering direct links in sparse graphs, the time complexity for learning GCNs is $O(|E|) \simeq O(|V|)$. We note that $|V|$ is the number of vertices and $|E|$ is the number of edges of an input graph. However, as higher-order paths connect distant nodes, its complexity becomes $O(|V|^2)$, we propose a more efficient mini-batch learning method. Incorporating higher-order paths can increase the relational signal by considering *nearby* but not directly linked nodes, however, it can also increase noise due to spurious connections as neighborhood size increases. To account for this and enable the model to *learn* which distant nodes are more relevant, we propose a novel attention mechanism based on *role equivalence*, a social network property that quantifies similarity among nodes based on their relational context. The attention mechanism is used to merge the multiple node representations learned from the set of high-order-based GCNs.

Moreover, network interactions are often changing and evolving over time in real-world domains. For example, users develop their connections to each other in social networks while communicating with others, which could be potentially useful for identifying their class labels such as political views and genders. Nonetheless, it is difficult to leverage the dynamics of temporal interactions for node classification. In particular, when the interaction edges are very sparse in each temporal snapshot or a node's neighborhood is biased toward a particular class label in different temporal snapshots, the use of temporal patterns may not be useful for predicting class labels. In these cases, static and aggregated patterns may help to offset these issues. Our proposed model can not only learn temporal interaction patterns but also model the aggregated neighborhood for node classification.

Finally, while graph representation models have increased the performance of many relational tasks such as node classification and link predictions, it compromises their ease of interpretation. Providing interpretable explanations is a crucial criterion when the embedding models are applied in time-critical and cost-intensive areas such as biology [12] and medicine [13]. Recently, similar research questions have been raised by researchers, and many explanation methods are proposed to provide explanations with salient features [14, 15] or make the models more transparent [16]. However, most of them have focused on providing explanations for predictive learning models on simple sequenced or grid inputs. Moreover, since graph representation learning methods such as DeepWalk [1], LINE [2], Struc2Vec [3], and PTE [4] are typically trained in an unsupervised and transductive fashion, the previous explanation tools is not directly applicable. Recently, while GNNExplainer [17] is proposed to provide explanations for Graph Neural Network models (e.g., GCN [5]) w.r.t. a single prediction, and it is still required to have a specified end-task and requires additional computationally heavy learning ($= O(|V|^2)$ in the worst case). Moreover, it has not been evaluated with real-world graphs. In this dissertation, we propose a novel explanation method for finding globally important nodes to the embeddings, which are learned from the aforementioned models. Our analysis and proofs reveal that node-level importance in DeepWalk could be measured by *bridgeness* under cluster-aware local perturbation, and we provide an efficient gradient-based method, which we call GRAPH-wGD, to approximate the *bridgeness*-based ranking. To evaluate our proposed methods, our graph neural network architectures (REGNN and TSGNet) consistently improve predictive performance on real-world datasets. Highly ranked nodes, which are found by our GRAPH-wGD, also have more impacts on (1) changing predicted probabilities for node classification and (2) assigning different local neighbors in the embedding space when we perturb them and re-learn low-dimensional node representations.

1.1 Problem Statement

In this dissertation, we aim to provide new neural network architectures and an explanation method for learned node representations in semi-supervised learning tasks. More specifically, we intend to either answer or provide insights to these questions:

1. With respect to exploiting local inferences with high-order paths in graph neural network for semi-supervised learning, we initially compute high-order (or k-th order) proximity matrices from the input data. The high-order matrices provide long-range path information that is often helpful for sparse graphs. However, the k-th order matrices could become quickly dense and label inference could be biased by increased noisy information. We would like to verify whether the

role equivalence among nodes can reduce the noise and evaluate whether our proposed architecture could be learned using a mini-batch training for better efficiency.

2. With respect to learning both temporal and static interaction patterns, we proposed a graph neural network architecture that jointly finds latent representations of nodes. In this work, how to learn these two different interaction patterns effectively is explored for enhancing graph neural networks.

3. Finally, we investigate how to provide global explanations by finding nodes who contributed the most to learned representations. We study whether nodes with high *bridgeness* have larger impacts on the learned node representations after perturbing the nodes and training the graph again. Moreover, we investigate how to approximate the ranking using the *bridgeness* to avoid high time-complexity on computing *bridgeness* for all nodes.

1.2 Main Hypothesis and Proposed Research

The goal of this dissertation is to verify the following hypothesis. (1) *The noise from high-order path-based Graph Neural Networks (GNNs) can be reduced by our proposed role equivalence attention layer, and its learning can be also improved by the importance sampling-based mini-batch training.* (2) *To learn both temporal and static interaction patterns effectively, a neural network architecture who includes GNN-based temporal encoder and static encoder on aggregated neighborhoods can enhance the existing static GNNs and temporal node embedding methods.* However, learned node representations are usually less interpretable than the previous relational learning models due to the lack of explanation methods. To address the problem, we also verify that (3) *Nodes with high bridgeness have larger impacts on the learned node representations under perturbation with respect to pair-wise distances and ranking of bridgeness can be approximated using a gradient-based method for better efficiency.*

1.2.1 Proposed Research

This dissertation is divided into 3 components.

1. The first part presents a role equivalence attention-based graph neural network architecture for higher-order label propagation.

2. The second part is to propose a graph neural network architecture that can jointly learn both temporal and static interaction patterns.

3. The third part shows the impact of high *bridgeness* on learned node representation under perturbation and proposes a gradient-based explanation method to efficiently approximate *bridgeness*-based ranking.

This document is organized as follows. First, we introduce the background of important concepts that are needed to discuss further ideas. Second, we propose a high-order path-based graph neural network architecture for exploiting local inferences via labels. Third, a GNN architecture is explained for learning both temporal and static interaction patterns. Lastly, we show a theoretical study and propose a novel gradient-based explanation method, which we call GRAPH-wGD, for finding globally important nodes to the embeddings, which are learned from node representation learning methods such as DeepWalk [1], LINE [2], Struc2Vec [3], and PTE [4].

2 BACKGROUND

2.1 Semi-Supervised Learning for Node Classification

In this dissertation, we consider the problem of classifying nodes in a graph, where labels are only available for a small subset of nodes. The main idea of semi-supervised learning is to leverage unlabeled nodes to improve prediction performance. Based on both labeled and unlabeled nodes, the problem of semi-supervised learning is defined as learning a classifier with or without attributes. This problem can be framed as graph-based semi-supervised learning, where label information is smoothed over the graph via some form of explicit graph-based regularization. The initial attempts are described in the following subsection.

2.1.1 Graph-based Semi-Supervised Learning

Previous graph-based semi-supervised node classification algorithms such as [8, 18–21] learn a model to predict class labels of a set of unlabeled nodes, V_U. They assume that nearby nodes are likely to have the same labels. Their loss functions are formulated as:

$$\mathcal{L}_{semi.classification} = \mathcal{L}(X, Y_L) + \lambda \mathcal{L}_{reg} \tag{2.1}$$

$\mathcal{L}(X, Y_L)$ is a supervised loss with respect to the features, X, and the known labels, Y_L. The second term is a loss for regularization with respect to the graph structure. For example, Label Propagation [18] and ICA [8] estimate labels of unlabeled nodes using the local inference by a local classifier or the propagation of neighboring nodes' information. ManiReg [19] explicitly has the loss function of SVM for the supervised loss. We show analytically how one of our proposing GNN architectures extends the label propagation [18] via high-order paths and attention—the relationship is described in Section 3.4.2. Joint prediction methods via optimization with local conditional models are also relevant works. They seek to use predictions of unknown node labels in their learning procedures (e.g., [9, 22]). The approaches utilize the whole network to simultaneously estimate model parameters while predicting labels of unknown nodes. For example, Pseudolikelihood Expectation Maximization ($PLEM$) [9] proposed an expectation maximization (EM) method to estimate parameters and perform predictions in an iterative fashion. $PLEM$ also utilizes Maximum Entropy constraints in the inference step to produce highly calibrated probability estimates.

2.1.2 Node Embedding Approaches

Graph embedding methods learn low-dimensional node representations of nodes in a continuous vector space. The idea generally shows better performance for node classification when the input graph is sparse. The aim is to learn a dictionary $Z \in \mathbb{R}^{|V| \times d}$ where d is a size of representation and $|V|$ is the number of nodes. In particular, the random walk-based models (e.g., Node2Vec [7]) exploit Skip-Gram architecture or negative sampling to maximize the posterior probability of observing a neighboring vertex in a random walk. They have shown usefulness for many relational learning tasks like node classification and link predictions. High-order path information also has been utilized in many relational embedding methods. For example, LINE [2] proposed a graph embedding algorithm that uses first-order and second-order proximity information. The algorithm proposed a new objective function to the two proximity information and showed promising results for node classification. Planetoid [6] also proposed a neural network architecture for semi-supervised learning with graph embeddings. They developed both transductive and inductive learning methods, but it is limited for modeling diverse relational properties such as structural similarity and high-order paths. Meanwhile, GraRep and Stuc2Vec were proposed to exploit the properties in order to learn latent representations. GraRep [23] attempts to learn low-dimensional representations of nodes with the SVDs of high-order matrices. However, it is not scalable for large-scale network data. NEU [24] is another recent work that uses high-order paths for node classification. Their approach is to enhance the trained node embedding representations using high-order path information. By simple post-updating steps, they achieved considerable enhancement. Struc2Vec [3] models a graph to reflect isomorphism among nodes and captures structural similarities by the Skip-Gram architecture. It is also computationally infeasible because it computes structural similarity using pair-wise degree sequences. Recently, VERSE [25] was developed to learn over many different similarities including structural similarity, at the same time using scalable sampling techniques.

2.1.3 Graph Neural Networks (GNNs)

In addition, deep learning architectures also have attracted a lot of attention. The prominent examples for node classification are Convolutional Neural Networks (CNNs) [26] and Graph Convolutional Networks (GCNs) [5]. Initially, CNNs were generalized for graphs in the spectral domain using Chebyshev polynomials which could be trained in a form of stacked neural networks [26]. Then, GCNs [5] simplified the spectral graph convolution using normalization techniques, which improved accuracy and efficiency over [26]. GCNs use the connectivity structure of the graph as a filter to perform neighborhood aggregation and construct low-dimensional representations of nodes. This can be understood as a special case of the message-passing framework [27], which is also called Graph Neural Networks (GNNs).

Again, GNNs also use the graph structure and node features to learn a representation vector of a node, h_v. Many of GNN architectures [5,10,26,28] follow a specific neighborhood aggregation strategy, and the representation of a node is iteratively updated by $a_v^{(k)}$ as below. After k iterations of aggregation, a node's representation captures the structural information within its k-hop network neighborhood, and self-representation of the node is combined using a COMBINE function. Formally, the k-th layer of a GNN is defined as:

$$\begin{aligned} a_v^{(k)} &= \text{AGGREGATE}^{(k)}(\{h_u^{(k-1)} : u \in \mathcal{N}(v)\}) \\ h_v^{(k)} &= \text{COMBINE}^{(k)}(h_v^{(k-1)}, a_v^{(k)}), \end{aligned} \tag{2.2}$$

where $h_v^{(0)}$ is set to an input feature matrix X_v, and $\mathcal{N}(v)$ is defined as a set of nodes adjacent to v.

The AGGREGATE and COMBINE functions are differently defined depending on the relational assumption of each method. For example, in GCN [5], mean-pooling is used for aggregating nearby information, and, in GraphSAGE [10], other strategies such as max-pooling and LSTM are used for updating the representation.

Recently, high-order path information was also incorporated into GNNs. HA-GCN [29] initially proposed to expand the GCN. It uses an element-wise matrix product instead of a linear transformation of GCN, and their experiments showed promising results on node classification.

2.1.4 Attention-based Graph Neural Networks

Graphs are often complex and noisy, so many researchers have incorporated the concept of "attention" [30] into semi-supervised classification. The attention mechanism allows a model to place more importance on task-relevant parts of the graph. In particular, message-passing frameworks [31–33] have been proposed to moderate the effects of neighboring nodes depending on their self-representation and local graph structure. Similar to GNNs, after k iterations of aggregation, the aggregation function of attention-based graph neural networks at k-th layer can be generalized as:

$$h_v^{(k)} = \text{ATTENTION\_AGGREGATE}^{(k)}(\{z(\alpha_{u,v}^{(k)}, h_u^{(k-1)}) : u \in \mathcal{N}(v)\}), \tag{2.3}$$

where $h_v^{(0)}$ is set to an input feature matrix X_v, and $\mathcal{N}(v)$ is defined as a set of nodes adjacent to v. The function z decides the final representation of u before aggregation, and $\alpha_{u,v}^{(k)}$ is used for assigning the weighted importance of each u to v. We note that the COMBINE function of attention-based graph neural networks is often integrated to the ATTENTION_AGGREGATE function. For example, GAT [31] proposed a self-attention-based graph neural network, which computes a weighted sum using the representation of edges. GAT at the k-th layer sets $\alpha_{u,v}$ and $z(\alpha_{u,v}^{(k)}, h_u^{(k-1)})$ as:

$$\begin{aligned} \alpha_{u,v}^{(k)} &= \frac{\exp\ (\mathrm{LeakyReLU}((a^{(k)})^T[W_{ATT}^{(k)} h_u^{(k-1)} || W_{ATT}^{(k)} h_v^{(k-1)}]))}{\sum_{m \in \mathcal{N}(v)} \exp\ (\mathrm{LeakyReLU}((a^{(k)})^T[W_{ATT}^{(k)} h_m^{(k-1)} || W_{ATT}^{(k)} h_v^{(k-1)}]))} \\ z(\alpha_{u,v}^{(k)}, h_u^{(k-1)}) &= \alpha_{u,v}^{(k)} W_{ATT}^{(k)} h_u^{(k-1)}, \end{aligned} \tag{2.4}$$

where $a^{(k)}$ is a trainable attentive vector at the k-th layer. We note that $W_{ATT}^{(k)}$ is a trainable variable for the attention layer and the aggregation function of GAT. However, the edge representation of GAT is a simple concatenation of the latent representation of the incident source and target nodes. Meanwhile, VAIN [33] also proposed a kernel-based attention mechanism for multi-agent modeling, and they exploited the similarity between nodes based on communication vectors (thus, attributes.) In the case of VAIN, they model the interaction among $|V|$ agents, and the output can be either be a prediction for every agent or a system-level prediction. $\alpha_{u,v}$ and $z(\alpha_{u,v}^{(k)}, h_u^{(k-1)})$ at the k-th layer of VAIN are defined as:

$$\begin{aligned} \alpha_{u,v}^{(k)} &= \mathrm{Softmax}(-|h_u^{(k-1)} - h_v^{(k-1)}|^2) \\ z(\alpha_{u,v}^{(k)}, h_u^{(k-1)}) &= \alpha_{u,v}^{(k)} e(h_u^{(k-1)}), \end{aligned} \tag{2.5}$$

where e is a communication embedding function for an agent u.

In Chapter 3, a new kernel-based attention approach is proposed to learn latent representations via high-order path information. While previous attention mechanisms have focused on moderating the relative importance of *direct* neighbors, our approach considers a larger set of neighbors (i.e., determined through high-order paths) and uses a notion of *role equivalence*. We compare ours to GAT [31] and show that our proposed attention mechanism offers significant improvement in node classification.

2.2 Interpreting Outputs of Machine Learning (ML) Models

2.2.1 Notion of "Importance"

Many researchers have suggested methodologies to interpret the outputs of their target ML models by finding the most contributed (or important) features and have used the concept of "importance" from various perspectives. The importance of features in ML networks can be understood from three different categories: degradation of prediction scores after perturbing the features, humans' participation, and design principles of data. We note that most of the works take non-graph inputs such as images and sentences and find which features (e.g., pixels or words) are contributed the most to the learned ML models. In this subsection, we briefly overview the previous methods in order to review how to define the notion of "importance".

Importance from Degradation of Prediction after Perturbing Importance Features Many previous methods [14, 34–38] define the concept of importance by leveraging the degradation of prediction after perturbing candidate features such as pixels and words. Their proposed importance function g could be generalized as below:

$$g(w_j|\mathrm{x}, c) = p(\hat{y} = c|\mathrm{x}) - p(\hat{y} = c|\dot{\mathrm{x}}_{w_j}). \tag{2.6}$$

$p(\hat{y} = c|\mathrm{x})$ is a predicted probability given the input feature x for a class label c, and $p(\hat{y} = c|\dot{\mathrm{x}}_{w_j})$ is the new prediction after the perturbation of a partial feature w_j. However, the predictions (e.g., soft-max outputs) are often not calibrated well as in [39]. For example, it overestimates the model's confidence when making predictions on inputs that fall outside the training distribution.

Importance from Human Participation The best way to define the concept of importance is to ask people to participate only if there are objective and specific criteria. In particular, defining important features through human participation has often been used for image classification and object detection tasks. For example, [40] used the ImageNet localization challenge to determine important pixels for identifying labels, and [41] evaluates deep neural networks with the number of unique objects, given by humans. [42] also employed the datasets which contain the label of image segments with the location provided by people's direct participation. However, this method is often complicated to say which nodes are more important when the input graph lacks domain knowledge.

Importance from Design Principles of Synthetic Data Generating synthetic data with important features specified by domain experts is also an excellent way to evaluate. For example, [16] is a recent interpretation method for finding the exact decision boundary to interpret a specific type of neural network. For assessing the algorithm, the authors used a synthetic input that is composed of two class labels. Their decision boundary was evaluated to compare the found set of linear boundary functions. Second, [43] also generates synthetic data to evaluate their interpretation methods by re-generating data under specific assumptions from the existing datasets. One example of their methods is to sampling a DNA sequence and to create another sequence dataset from the existing DNA datasets. They assume that a sequence of motifs is essential to identify a protein. Similarly, [44] made a set of rules to assign labels for generating synthetic data. For instance, if pixels at all corners have the same colors, they put "1" s as labels. Another example can also be found in [45]. They generated synthetic data which are conditionally independent to n-dimentional gaussian features. While many ideas have been proposed for non-graph inputs, however, there is little work done for relational machine learning models that take graph inputs.

2.2.2 Interpretation Methods for Machine Learning Models

A variety of relevant studies can also be analyzed in terms of methodology for interpretation. We've divided it into three main categories, and they are as follows: hidden neuron analysis, model mimicking methods, and local interpretation methods. We also note that they mainly aim to understand the outputs of ML models that take non-graph inputs.

Hidden Neuron Analysis The hidden neuron analysis allows us to interpret a trained deep neural network by model-specific methods. Yosinski et al. [46] attempted to visualize the learned features for convolutional neural networks and proposed a regularized optimization to produce a better interpretation. Cao et al. [47] showed the activation status using attention on its target objects by a feedback loop. Li et al. [48] visualized the compositionality of clauses by analyzing the outputs of hidden neurons in a neural model for natural language processing.

Model Mimicking Methods Approaches [16, 49–51] for mimicking neural networks help to build a transparent model. Due to the transparent model and interpretable parameters, the approaches give ways to explain predictions while preserving their classification accuracy. Ba et al. [49] proposed a model compression method to train a shallow mimic network using the training instances labeled by one or more deep neural networks. Hinton et al. [50] proposed a knowledge transfer method that distills the knowledge of a large neural network by training a relatively smaller network. This can be seen as replicating the prediction probabilities of the original large network. Zhu et al. [51] built a forest model on top of a deep feature embedding network. The forest model provides more interpretability, but it is still difficult to understand representation learning with respect to raw data. The above models are more straightforward to interpret than deep neural networks. However, due to the reduced model complexity, there is no theoretical guarantee for verifying how much the interpretation is trustable.

Local Interpretation Methods The local interpretation methods compute and visualize the important features for an input instance by analyzing the predictions of their local perturbations. For example, LIME [14] learns a local decision function by perturbing potentially important features. Their subsequent work [52] expands the LIME by providing sufficient conditions to provide more precise interpretation. SHAP [15] also generalizes LIME by exploiting the concept of Shapley value in game theory and proves that there is a unique solution under desirable conditions. The local interpretation methods generate an insightful individual interpretation for each input instance. However, they have not applied to relational domains, and the naive perturbation would take too many resources.

Interpretation Methods for Graph-based Machine Learning Models The previous methods mainly aim to provide explanations to the ML models who take non-graph inputs. To understand outputs from graph-based ML models, recently, a few existing studies were proposed. First, GNNExplainer [17] has been proposed to provide local explanations for graph neural network models (GNNs) by learning soft-masks. For example, the local explanations here represent important edges or features which are contributed the most to predicting labels of a specific node. However, the method does not have the capability of providing global explanations for the node embedding itself. GNNs are also interpreted in [53] as well using contrastive gradient-based (CG) saliency maps [54], Class Activation Mapping (CAM) [55], and Excitation Backpropagation (EB) [55], and their variants. However, they also focus only on the specific graph neural networks, and could not be extended to node embedding models or others. Taxonomy induction method [37] and XGNN [56] are also relevant to this work, but they interpret node embeddings only by identifying clusters in the embedding space or providing a model-level interpretation with a graph generation, respectively.

2.2.3 Comparison to Related Work

We summarize the theoretical and empirical contributions of this dissertation. Table 2.1 shows how this dissertation is related to the existing works such as Label Propagation (LP) (e.g., [57]), Probabilistic Approach (e.g., [22]), Node Embedding Models (e.g., [1,58]), and GCN [5]. In the Table, "Learning static graph" and "Learning temporal graph" represent which methods are used for learning an aggregated graph and a set of temporal graphs, respectively. Collective inference and high-order path mean whether the method supports collective inference and leverages high-order paths, respectively. Explainability indicates how the output of the corresponding work can be explained. While LP and Probabilistic Approach in the table are not based on representation learning, however, they support collective inference for static graphs. Node embedding methods show promising results in many relational learning tasks, but most of them are limited for learning temporal graphs. Some models (e.g., [59]) attempted to learn temporal graphs for node classification, but they model specific temporal patterns only.

In this dissertation, first, our proposed REGNN expands the GCN to exploit the labeled information with collective inference via high-order paths. In particular, the architecture is useful when attributes are not given, and it is theoretically related to Label Propagation [57]. Second, our TSGNet proposed a graph neural network architecture to additionally learn temporal interactions. The two different encoders are learned jointly and could overcome temporal sparsity or ambiguities on the aggregated

Table 2.1.: Summary of contributions: comparison to related work

	Learning Static Graph	Learning Temporal Graph	Collective Inference	High-Order Path	Explainability
LP (e.g., [57])	✓	×	✓	×	-
Prob. Approach (e.g., [22])	✓	×	✓	×	-
Node Embedding (e.g., [1])	✓	△	×	×	**GRAPH-wGD**
GCN [5]	✓	×	×	×	-
REGNN	✓	×	✓	✓	-
TSGNet	✓	✓	×	×	-

graph. Third, our GRAPH-wGD provides global explanations to Skip-gram-based node embedding models, and the theoretical relationship to *bridgeness* is provided.

3 ROLE EQUIVALENCE ATTENTION FOR LABEL PROPAGATION IN GRAPH NEURAL NETWORKS

3.1 Motivation

Prior work on semi-supervised learning in graphs has typically defined relational features via aggregation over the information of neighboring nodes, and then unknown class labels are inferred iteratively using approximate inference algorithms (e.g., Gibbs sampling [8] and composite-likelihood EM [9]). Meanwhile, recent graph neural network models [5,10,11,60–62] learn relational features using LSTM and CNN-like architectural components. In particular, graph convolution networks (GCNs) [5] provide a powerful deep neural network architecture, which exploits convolution operators for learning based on nearby neighbors' attributes. The graph convolution operator sums attributes of all the neighbors of nodes with a linear transformation. By stacking multiple layers, it learns a low-dimensional representation of each node. However, GCNs mainly aims to learn via neighbor attribute patterns—they are not typically used in partially-labeled graphs with few attributes, where *local inference* is needed during learning, so that patterns in neighbor class labels can also be used in the model. In this section, we propose another graph neural network architecture for semi-supervised learning in this setting, and we call the architecture, REGNN.

Our proposed approach initially computes high-order (or k-th order) proximity matrices from the input data. The high-order matrices provide long-range path information that is often helpful for sparse graphs. The k-th order matrices contain statistics from k-step random walks in the graph, so they are useful to learn complex patterns from different levels of local neighborhoods. High-order paths have been utilized recently for many relational learning tasks [23, 29]. This includes variants of GGN [29], but the method simply concatenates representations from different GCN layers and learns the architecture for node classification. As such it is likely still dependent on the availability of observed attributes, since it was not designed for semi-supervised learning contexts. In contrast, we focus on semi-supervised learning in partially labeled graphs and how to moderate the high-order paths with an attention mechanism in the architecture.

The idea of attention mechanisms was originally developed for machine translation [30, 63] in order to consider the weighted average of all the previous states for deciding the next state. Similarly, attention mechanisms for graphs [31, 32, 64] have been proposed to moderate the effects of neighboring nodes when learning low-dimensional representations. However, these prior methods use relatively simple attention mechanism based primarily on the neighbor/edge representation, e.g., simple simply concatenating the representations of both source and target nodes [31] or considering direct edges only for adjusting the weights [32]. Our key insight is to develop an attention mechanism that can be moderated by the *relationship* between the two nodes, specifically their *structural role equivalence*.

Node equivalence measures are often used in social network analysis to quantify the similarity between a pair of nodes based on their link structure. Two nodes are regularly equivalent if they have the same neighbors. Meanwhile, they are structurally equivalent if their neighbors have the same roles (see Section 3.4.1 for definitions). The concepts were originally proposed in social science [65–68], and they have recently been applied to graph embedding methods [3, 25] to improve classification and clustering.

We conjecture that attention mechanisms based on role equivalence will more effectively exploit long-range path information in the network, by allowing the model to identify the more useful distant neighbors during aggregation. However, it is computationally intensive to use role equivalence for attention over high-order paths in a semi-supervised learning context, because the density of the matrices increases with k. To address this issue, we outline a mini-batch training approach that uses importance sampling to downsample possible neighbors on each epoch.

3.2 Motivating Examples

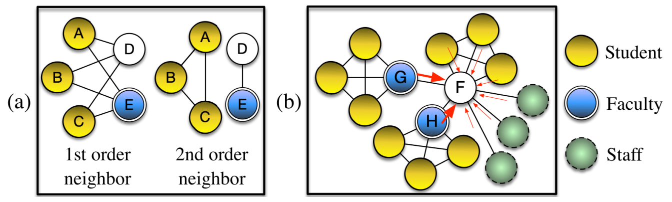


Figure 3.1.: Examples of capturing roles: (a) by High-order paths (b) by Similarity-based attention

Figure 3.1a-b show examples that high-order path and attention can help to capture roles in the label propagation scenarios. Each node and edge indicate a user and an interaction during a semester, respectively. Note that colors represent class labels, yellow for student, blue for faculty, and green for staff. In Figure 3.1a, we are trying to predict the label of a user D, who is a faculty. When we use just direct neighbors, it is not possible to predict the true label of node D by label propagation. However, as the high-order paths from 2^{nd} order neighbors are considered, the label of D could be successfully predicted. Like this example, high-order paths are potentially useful to learn the underlying hierarchical roles such as advisor-student and admin-member relationships in a citation network and a University group on Facebook, respectively. Meanwhile, if a user is surrounded by nodes that have diverse class labels, it often misleads its prediction just by magnitudes of nearby labels. In this case, if latent representations, which are trained in advance, are similar, we can put more importance when considering neighbors. In Fig. 3.1b, we are trying to predict the label of node F, who is also a faculty member. Although the node F has more students or staffs as neighbors, node G and H are likely to have similar representations to the representation of node F in the latent space, so that they can have more importance when aggregating. In this chapter, to address the first case, label propagations are attempted via high-order paths and latent representations are learned. Then, additional propagation is performed in an attention layer.

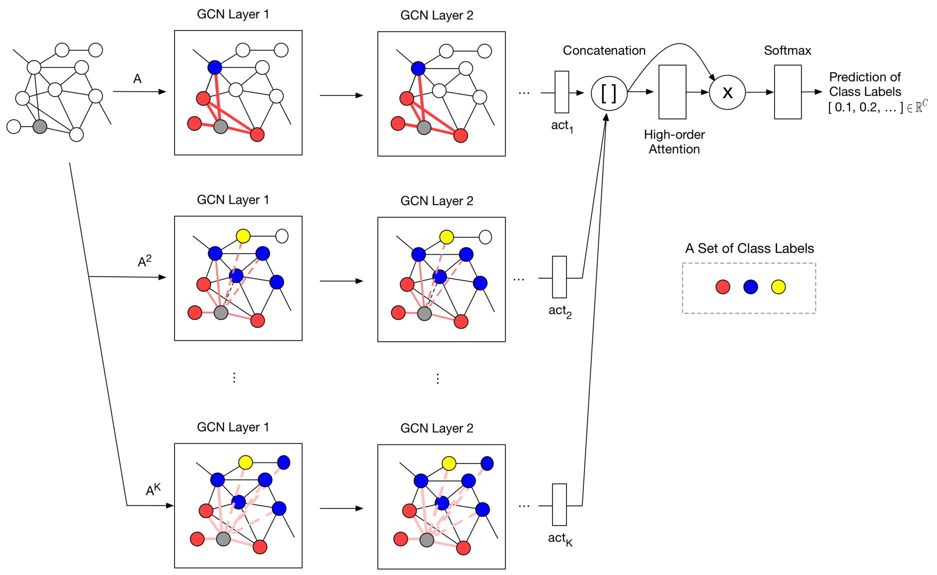


Figure 3.2.: Architecture for *REGNN*

3.3 Notation

We have an undirected graph $G = (V, E)$ with $V = |V|$ vertices and $E = |E|$ edges where each vertex v_i is associated with a label $y_i \in \{0, 1\}^C$. A is an adjacency matrix of the graph G. We assume the number of classes C is known, and all classes are present in the labeled data. For a node $v \in V$, $\mathcal{N}(v) = \{u|(v, u) \in E\}$ is the set of its neighborhoods. V is composed of V_L (labeled vertices) and V_U (unlabeled vertices). Our goal is to estimate class labels of V_U from class labels of V_L and A, which is a transductive learning setting.

3.4 REGNN for Node Classification

In this section, we present the major components of our proposed model REGNN. There are two key components to the architecture: (1) a set of k-th order Graph Convolution Network layers, and (2) a self-attention layer. First, k-th order graph convolution network layers learn the complex neighborhood by exploiting not only direct edges and but also implicit relationship among nodes from k-order proximity information. As in Figure 3.2, $K \times M$ convolutional operators learn labels in local neighborhoods, where M refers to the number of GCN layers and K is the number of high-order matrices, which is pre-selected. Colors represent the most probable class labels, which are inferred by convolutional operators. Convolutional operators learn the labels jointly using high-order path information, and predict the label of unlabeled nodes (e.g. a grey node in the figure.) Then the representations are concatenated and aggregated by the weighted sum using a self-attention mechanism. Our high-order path attention layer determines the final representations for predicting class labels. The details of these components are described below.

3.4.1 REGNN Architecture

Given an undirected graph $G = (V, E)$, where V is a set of vertices and E is a set of edges. A is an adjacency matrix of G. V is composed of V_L (labeled vertices) and V_U (unlabeled vertices). Y_L is constructed as a $|V| \times C$ class label matrix. For each labeled node $i \in Y_L$ with class label $y_i = c$, we set $Y_L[i, c] = 1$ and $Y_L[i, \cdot] = 0$ otherwise. For example, node i is labeled as 1, then $Y_L[i] = [0, 1, 0, 0, 0, ..., 0]$. If node j is in V_U, we set $Y_L[j, :] = 0$. This Y_L will be fed to our REGNN with the adjacency matrix A. Thus, the goal of REGNN is to estimate the class labels of V_U from Y_L and A.

k-th Order Graph Convolution Network layers Motivated by the GCN [5], we propose k-th order Graph Convolution Network layers. For the layers, we make use of adjacency matrices, $A, A^2, ..., A^K$, which have different orders. A^k is obtained by the k-times multiplication of the adjacency matrix A. In other words, the (i, j) entry of it k-th product A^k is the number of k-hop paths from i to j. With these high-order adjacency matrices A, we can define a K-th order convolution operator as follows. The node representations in layer m with an adjacency matrix A^k are formulated as

$$H_k^{(m+1)} = \text{ReLU}\left(\hat{A}^k H_k^{(m)} W_k^m\right), \tag{3.1}$$

where

$$\hat{A}^k = min\left(\hat{D}_k^{-1/2}\left(A^k + I\right)\hat{D}_k^{-1/2}, 1\right) \tag{3.2}$$

In Equation (3.1), $H_k^{(1)} = Y_L$, and Y_L represents class label inputs. W_k^m is a trainable weight matrix for the m-th layer in the k-th order GCN, and $\hat{D} = \sum_j \hat{A}_{ij}$. The symmetric normalizing trick in Eq. (3.2) takes the average of neighboring nodes' representation from each of high-order adjacency matrices. Again, Y_L represents label inputs from training data. Note that $W_k^m \in \mathbb{R}^{C\times C}$, which means that propagated labels are transformed by the matrix multiplication.

When the representation of the last GCN layer, $H_1^{(M+1)}$, is additionally passed through another softmax function to consider the labels on the direct edges more, performance tends to improve. In practice, we find this idea improves the classification accuracy for REGNN as well, so in Figure 3.2, act_1 we use softmax(ReLU). In $[act_2, .., act_k]$ we use the ReLU representation directly.

Concatenation Layer Outputs from the previous high-order GCNs are concatenated before they are fed into a self-attention layer. There are K outputs, one from each of the high-order GCNs: H_1^{M+1}, ..., H_K^{M+1}. These are concatenated along with the node ID. Let $q_i^k \in \mathbb{R}^C$ be a latent representation of node i from H_k^{M+1}. Thus, $H_k^{M+1}[i,:] = q_i^k$. After the concatenation, q_i^{concat} is:

$$q_i^{con} = \overset{K}{\underset{k=1}{\|}} q_i^k. \tag{3.3}$$

Self-Attention Layer Another important part of our proposing architecture is the self-attention layer. Our self-attention layer measures the degree of *role equivalence* among nodes to place more importance on structurally similar neighbors. First, the mathematical definitions of structural equivalence [66] and regular equivalences [65, 67] are given below.

Definition 1. *(Structural Equivalence) A pair of nodes u and v is* structurally *equivalent, if the neighbors of node u and v are the same. Thus, u and v are structurally equivalent if and only if $\mathcal{N}(u) = \mathcal{N}(v)$.*

Definition 2. *(Regular Equivalence) A pair of nodes u and v is* regularly *equivalent if the roles of their neighbors are the same. Let $r(i)$ be the role of node i. Thus, u and v are regularly equivalent if and only if $\{r(i) \mid i \in \mathcal{N}(u)\} = \{r(j) \mid j \in \mathcal{N}(v)\}$.*

In other words, regular equivalence states that nodes play the same role if they have similar connections to nodes of other roles [68, 69]. There might be many valid ways of grouping nodes into equivalence role sets for a given graph, and regular equivalence is often defined recursively.

Based on the above definitions, we can approximate the notion of regular equivalence based on roles in latent space.

Definition 3. *(Role Equivalence in latent (embedding) space) A pair of nodes u and v are* role *equivalent in latent space if their set of neighbors in latent space are the same. If neighbors are defined by distance in latent space, then u and v will have the same neighbors if their representations are equal. Let $f(i)$ be the latent representation of node i. Thus, u and v are role equivalent in latent space if and only if $f(u) = f(v)$.*

According to Definition 3, we propose an attention layer based on role equivalence among nodes. The intermediate representations of the last high-order GCN layers is used for defining the role, thus $f(i) := q_i^{con}$.

In this layer, by considering role equivalence, we can incorporate structural information into node classification. To measure the degree of role equivalence, we additionally define a quantitative measure of role equivalence in latent space with $\mathcal{RE}\big(f(i), f(j)\big)$, and use it in the attention layer below.

Our self-attention layer takes inputs from the concatenation layer and produces a new vector $q'_i \in \mathbb{R}^{K \cdot C}$ as:

$$q'_i = \mathrm{ReLU}\left(\sum_{j \in \mathcal{N}(i)} \mathcal{RE}\big(q_i^{con}, q_j^{con}\big) \cdot q_j^{con}\right), \tag{3.4}$$

where $\mathcal{RE}\big(f(i), f(j)\big) = (1/Z)e^{\beta cos\big(f(i),f(j)\big)}$, cos refers to cosine similarity, $Z = \sum_{j\in\mathcal{N}(i)} e^{\beta cos\big(f(i),f(j)\big)}$, and β is a variable that moderates attention (which we estimate during learning). Note that we do not consider self-loops for computing the similarity. The $\mathcal{RE}\big(f(i), f(j)\big)$ models how close to role equivalent node j is to node i (i.e., if the latent representations are unit vectors, then the two are equal when their cosine similarity is 1).

Final Softmax Layer To predict class labels of nodes, a final softmax function is used. Here, $\hat{y}_i$ is the output of the softmax function, and each dimension of the $\hat{y}$ represents the predicted probability of the corresponding labels for the class given inputs, $W_{\text{final}} \in \mathbb{R}^{K\cdot C\times C}$:

$$\hat{y}_i = \text{softmax}\left(W_{\text{final}}\vec{q'}_i + b_{\text{final}}\right).$$

For learning, like the original GCN, we use categorical cross-entropy as a loss function at the final layer.

$$\mathcal{L}_{REGNN}(L, Y) = -\sum_{V_L}\sum_{j=0}^{C-1} y_j log(\hat{y}_j) \tag{3.5}$$

In Eq. (3.5), C is the number of class labels. Since all activation functions are differentiable, learning is simply done via back-propagation. During the back-propagation, all Ws (W_k^m and W_{final}) and β are trained. To predict class labels for unlabeled nodes V_U, the $\hat{y}_i$ will be used for predictions.

3.4.2 Relationship to Label Propagation

In this section, we provide additional analytical characterization of REGNN, compared to Label Propagation [57].

Label Propagation (LP) Assume that $Y_L \in \mathbb{R}^{|V|\times C}$ is a label input matrix. Same as the previous notion of Y_L, for each labeled node $i \in V_L$ with class label $y_i = c$, we set $Y_L[i, c] = 1$ and $Y_L[i, .] = 0$ otherwise. If node j is in V_U, we set $Y_L[j, :] = 0$. Let $\widehat{\mathbf{Y}}$ be a prediction matrix, and $\widehat{\mathbf{Y}}[i, :]$ for each node $i \in V_U$ will be used for actual prediction. The prediction is from $\arg\max_j \widehat{\mathbf{Y}}[i, j]$. According to [57], the prediction will converge as in the equation below. α is a parameter in (0, 1) and specifies the relative amount of the information from its neighbors and the initial label information.

$$\begin{aligned}\widehat{\mathbf{Y}}_{LP} &= (I - \alpha A')^{-1} Y_L \\ &= (I - \alpha(I - L))^{-1} Y_L \\ &= ((1-\alpha)I + \alpha L)^{-1} Y_L\end{aligned} \tag{3.6}$$

Here A' is a normalized adjacency matrix of A and L is a normalized Laplacian matrix. We can refer to the eigen-decomposition of the normalized Laplacian matrix as $L = \Phi\Lambda\Phi^{-1}$. The Laplacian matrix, L, can be modified using the frequency response [70] as $L' = \Phi p(\Lambda)\Phi^{-1}$ where $p(\cdot)$ is called the frequency response function of the graph. $p(\Lambda)$ can further be written as $diag(p(\lambda_1), ..., p(\lambda_n))$. The graph L' is linear shift-invariant, if and only if there exists a function $p(\cdot) : \mathbb{R} \to \mathbb{R}$. The $\widehat{\mathbf{Y}}_{LP}$ can then be reformulated from the perspective of eigen-decomposition as below. At last, the $\widehat{\mathbf{Y}}_{LP}$ can be composed as a function (F) of the frequency response, $p_{\mathrm{LP}}(\Lambda)$, and an input matrix, Y_L.

$$\begin{aligned}\widehat{\mathbf{Y}}_{LP} &= (I - \alpha A')^{-1} Y_L \\ &= \Phi((1-\alpha)I + \alpha\Lambda)^{-1}\Phi^{-1} Y_L \\ &= F(p_{\mathrm{LP}}(\Lambda), Y_L)\end{aligned} \tag{3.7}$$

In this $\widehat{\mathbf{Y}}_{LP}$, $p_{\mathrm{LP}}(\lambda_i)$, the frequency response function of LP, is equal to $\frac{1}{(1-\alpha)+\alpha\lambda_i}$.

Graph Convolution Neural Network (GCN) Similarly, we can reformulate the high-order GCN of REGNN. Again, GCN take inputs from $\hat{A} = A + I$, $\hat{D} = D + I$, and $\hat{L}$ = Laplacian matrix of $\hat{A}$. Its normalized Laplacian matrix $\hat{L}_s$ can also be composed as below.

$$\begin{aligned}
\hat{A} &= \hat{D}^{-1/2}\hat{A}\hat{D}^{1/2} \\
\hat{A} &= I - \hat{D}^{-1/2}\hat{L}\hat{D}^{1/2} = I - \hat{L} \\
\hat{L} &= \Phi\hat{\Lambda}\Phi^{-1}
\end{aligned} \tag{3.8}$$

Using the above notation, we can reformulate two-layered GCN as:

$$\begin{aligned}
\widehat{\mathbf{Y}}_{GCN} &= \hat{A} \text{ ReLU } (\hat{A}Y_L W^0)W^1 \\
&\approx \hat{A}(\hat{A}Y_L W^0)W^1 \\
&= \hat{A}^2(Y_L W^0)W^1 \\
&= (I - \hat{L})^2(Y_L W^0)W^1 \\
&= (\Phi(I - \hat{\Lambda})\Phi^{-1})^2(Y_L W^0)W^1 \\
&= \Phi(I - \hat{\Lambda})^2\Phi^{-1}Y_L(W^0W^1) \\
&= F(p_{\mathrm{GCN}}(\hat{\Lambda}), Y_L)(W^0W^1).
\end{aligned} \tag{3.9}$$

Here $p_{\mathrm{GCN}}(\hat{\lambda}_i) = (1 - \hat{\lambda}_i)^2$ and GCN with k-th order matrix uses $p_{\mathrm{GCN(k)}}(\hat{\lambda}_i) = (1 - \hat{\lambda}_i)^{2k}$. This indicates that the high-order GCN suppresses the small eigenvalues around 1 more. In addition, when LP [57] uses the following frequency response function, $p(\lambda_i) = (1 - \lambda_i)^2$ with two linear transformations, the new $\widehat{\mathbf{Y}}'_{LP}$ would be same as $\widehat{\mathbf{Y}}_{GCN}$. Thus, $\widehat{\mathbf{Y}}_{GCN} \approx \widehat{\mathbf{Y}}_{LP'}(W^0W^1)$. Similarly, when the response function, $p_{\mathrm{GCN(k)}}(\hat{\lambda}_i)$, is used for LP, it could also approximate the GCN with the corresponding k-order paths. As a result, we can see that our REGNN learns outputs from different response filters jointly and aggregates the neighbor's representations using role equivalence.

3.4.3 Importance Sampling for Scalability

To calculate Equation 4.7, REGNN needs to compute the representation of all the nodes together. Batch algorithms cannot handle large-scale datasets because their slow convergence and difficulty to fit the whole graph in GPU memory. In addition, due to the lack of dynamic graph configuration capabilities in Tensorflow-like deep learning libraries, we have to create a $|V| \times |V|$ pair-wise dense matrix to compute the self-attention score. Specifically, even though the attention mechanism only considers neighboring nodes (i.e., $O(|E|)$), without a dynamic data pipeline construction in Tensorflow, we are required to use a static $O(|V|^2)$ construction. This is what limits the scalability of edge attention-based algorithms like [31, 33, 64].

To overcome the limitation, we propose an efficient sampling-based learning. First, we note that in the full REGNN:

$$(\hat{A}^k H_k^{(m)})_u = |V| \sum_{v=1}^{|V|} \frac{1}{|V|} \hat{A}^k[u, v] H_k^{(m)}[v, :]. \tag{3.10}$$

Importance sampling was originally proposed in FASTGCN [71]. We can use a similar approach to approximate Eq.3.10 with $|S|$ samples, $v_1, ..., v_s \in V$ for node u as follows:

$$(\hat{A}^k H_k^{(m)})_u \approx \frac{|V|}{|S|} \sum_{v_s \sim q(v)} \frac{1}{q(v_s)} \hat{A}^k_{red\_gcn}[u, v_s] H_k^{(m)}[v_s, :] \tag{3.11}$$

with the importance distribution $q(v) = ||\hat{A}[:, v]||^2 / \sum_{v' \in V} ||\hat{A}[:, v']||^2$. The $|S|$ samples $\{v_s\}$ are used in all k-order GCN layers as well as the attention layer to update weights as in Algorithm 1. At every epoch, all nodes are divided to create a batch, B. B is used for reducing the dimensionality of $\hat{A}^k$ and $\hat{A}^1$. Instead of $\hat{A}^k \in \mathbb{R}^{|V| \times |V|}$ for computing Equation 3.1-3.2, we use the reduced adjacency matrices $\hat{A}^k_{red\_gcn}$ for all k, which corresponds to $\mathbb{R}^{|V| \times |S|}$. Similarity, a reduced matrix $\hat{A}_{red\_att} \in \mathbb{R}^{|S| \times |S|}$ is used for identifying neighbors for Equation 3.4. This can help to avoid storing all dense k-order matrices during training and greatly reduces the computational burden. The reduced adjacency matrices are induced each epoch during training.

Algorithm 1: REGNN's mini-batched training (one epoch)

```
For each vertex $u$, compute sampling probability $q(u)$
for each batch, $B$ do
  Sample $|S|$ vertices, $v_1, ..., v_s \in V$ according to distribution $q$ from node set $B$
  Induce $\hat{A}^k_{red\_gcn} = \hat{A}^k[:, S]$ for all $k$
  Induce $\hat{A}_{red\_att} = \hat{A}^1[S, S]$
  Update $W^l_k$, $W_{final}$, $b_{final}$, and $\beta$
end for
```

Table 3.1.: Comparison of various semi-supervised learning algorithms in the transductive setting

Methods	Local-Inf.	Scalable	H-Order	Reg.Equ.
ICA [8]	✓	×	×	×
Node2Vec [7]	×	✓	×	×
Struc2Vec [3]	×	×	✓	✓
GraRep [23]	×	×	✓	×
NEU [24]	×	✓	✓	×
GCN [5]	✓	×	×	×
HA-GCN [29]	✓	×	✓	×
GAT [31]	✓	×	×	×
REGNN	✓	✓	✓	✓

3.4.4 Complexity Analysis

When the batch size $|B|$ is considered, the time complexity of learning REGNN (before importance sampling) is $O(|B||E|C^2 + |B||E|C)$, where C is the number of class labels, and $|V|$ and $|E|$ are the numbers of nodes and edges in the graph, respectively. $|B| = |V_L|$ before importance sampling. This complexity is on par with GCN and GAT. We note that $|B|$ is chosen using validation node sets.

After we apply importance sampling, the time complexity is $O(|S||E_S|C^2+|S||E_S|C)$. Here $|S|$ is the size of the sample set and is usually chosen from 32 to 256, so $|S| << |V|$. E_S is the set of edges among the sampled $|S|$ vertices. Again $|E_S| << |E|$.

Regarding space complexity, we note that all k-order matrices are preprocessed and do not need to be stored in the main memory. Then, $(|V| \times |S|)$ and $(|S| \times |S|)$ matrices are indexed depending on the sample S. Therefore, despite the density of original k-order matrices, the space complexity of learning depends on the number of edges in $\hat{A}^k_{red\_gcn}$, which is at most $O(|V||S|)$.

3.5 Properties of REGNN Compared to Related Work

Tabel 3.1 categorizes our model and previous approaches based on five desirable properties for node classification tasks. The properties are defined as follows:

Local Inference (Local-Inf.): Able to combine the estimated class labels of unlabeled nodes for learning. This property helps the model to learn homophily patterns using the entire graph structure in order to make predictions. ICA uses a local classifier to make predictions at each iteration, and GCN/GAT sums nearby information with a linear transformation.

Scalable: Able to process large-scale networks. We assume time complexity larger than $O(|V|^2)$ is not scalable. GraRep is computationally heavy due to eigen-decomposition. GCN and GAT are corresponding to $O(|V||E|)$ when batch learning is used. When we assume $|E| = O(|V|)$, which is a common definition for a sparse graph, it still corresponds to $O(|V|^2)$

High-Order (H-Order): Exploits high-order path information. NEU, GraRep, and REGNN uses k-order adjacency matrices, and Struc2Vec also learns degree sequences among nodes using k-hop proximity.

Regular Equivalence (Reg.Equ.): Leverages regular/role equivalence to boost performance. This property additionally enables the model to learn roles and positions in the embedding space. REGNN uses role equivalence to adjust the relative importance of neighbors at the attention layer, and Struc2vec is a node embedding algorithm that forces nodes to have similar embeddings when structural distances of nodes are similar.

Tabel 3.1 shows how the properties of REGNN compare to previous state-of-the-art methods.

3.6 Experimental Evaluation

3.6.1 Data

We use five real-world network datasets for evaluation. Table 3.2 reports brief statistics for each network. In the table, $|V|$ and $|E|$ are the numbers of vertices and edges in each dataset. Cora, Citeseer, PubMed, and NELL are from [6,8]. For NELL data, the ratio of labeled nodes is 10 percent. The Facebook network was scraped from a University group. In particular, each user (node) of the Facebook dataset is associated with political views for their class labels. An edge is formed when a user writes a post to his or her friend's wall. We did not count self-posts. The data were randomly sampled to make its class labels' proportion to 50/50 from [9].

Table 3.2.: Data Statistics

Dataset	Nodes ($\|V\|$)	Edges ($\|E\|$)	Classes (C)
Cora	2,708	5,429	7
Citeseer	3,327	4,732	6
Facebook	4,038	65,794	2
PubMed	19,717	44,338	3
NELL	65,755	266,144	210

3.6.2 Experimental Setup

LR Logistic regression is performed using neighbor vectors with L1 regularization. This allows us to compare how GCN's convolutional operators improve performance.

GhostEdge GhostEdge [21] is a relational classifier which exploits personalized PageRank to create additional implicit edges. For this experiment, we use GhostEdgeNL, and it decides labels of nodes based on the class labels of neighboring nodes after quantifying node proximity. This was simple but effective on node classification.

ICA ICA [8] is one of the representative approximate inference algorithms for semi-supervised learning. SVM is also used for its local classifier and takes labels of neighbor nodes as inputs. The learning was stopped when all class labels had stabilized.

GCN GCN [5] is an original graph convolution network architectures, which is proposed for semi-supervised learning. To evaluate whether it could be used for label propagation, we also tested this architecture with label inputs, Y_L, and we call it as

GCN(Label) in the later sections. This is to compare how K-th Order inputs and attention layer can increase the performance.

Node2Vec Node2Vec [7] is a graph embedding algorithm for node classification. For learning its parameters, we set $d = [32, 64, 128]$, $r = 10$, $l = 80$, $k = 10$, and p and q were searched over [0.5, 1, 2].

GraRep GraRep [23] takes into account high-order matrices to learn node embedding. For learning its parameters, we set $d = [32, 64, 128]$, and K was searched over [2, 3, 4].

NEU NEU [24] is another recent work which learns high-order proximity for node classification. In this experiment, Node2Vec is used for its base algorithm, and we set $\lambda_1 = 0.5$, $\lambda_2 = 0.25$, T=3 for its hyper-parameters. These parameters showed the best performance for NEU [24]'s datasets including Cora, so we followed the authors' recommendation. This puts more importance on the direct edges using λ_1. Hyper-parameters of the base algorithm were searched as the Node2Vec did above.

Struc2Vec Struc2Vec [3] is to learn node representations in order to capture structural identity. To see its maximum performance, we do not utilize their optimization options. Other parameters for Skip-Gram architecture are the same as Node2Vec's ones.

VERSE VERSE [25] is a scalable embedding algorithm that learns many different similarities among nodes including structural similarity. For learning its parameters, we set $d = [32, 64, 128]$, and other parameters were chosen as what the authors used. Personalized PageRank is exploited for the similarity function in VERSE.

GAT GAT [31] is a recent attention-based neural network architecture for graphs. The architecture exploits another self-attention mechanism to model the complex neighborhood on the graph. This is to compare how our attention layer works. We had tested multi-head attention (K=1or 2) with averaging and concat options. In this algorithm, we did use importance sampling. When we tested GAT without importance sampling, the performance was almost similar.

HA-GCN Authors of HA-GCN [29] proposed a K-th order convolution operator under the GCN framework. For the node-centric learning like our problem definition, HA-GCN without the adaptive filtering showed better accuracy, so we followed the experimental setting.

3.6.3 Evaluation Methodology

We train the models on training/validation sets and report results on the test set. Every result we report is from the average of 10 trials using randomly shuffled node sets. Note that the entire graph is known before learning; 10% of node labels are used for testing and 10% of the remaining labels are leveraged for validation. The number of nodes of training is varied depending on experimental conditions. In the REGNN, the class labels from labeled nodes are only utilized for learning architecture. For all neural network models, the max epochs are set to 1000, and if the accuracy on the validation dataset does not increase during 100 epochs, learning stops. We also use dropout regularization (0.2). For optimization, we use the *adam* optimizer to update variables. For GCN, and HA-GCN, the number of hidden nodes is searched over [8, 16, 32, 64]. In the case of GAT, all experimental conditions are the same as REGNN except the attention layer. In order to choose learning parameters, after searching over [0.001, 0.005, 0.01, 0.03], 0.01 was used for learning our models. For selecting K-th order for REGNN and high-order neural network baselines, [2, 3, 4, 5] are considered and selected using the loss scores from the validation dataset. The number of GCN layers for all baselines and REGNN is 2. For importance sampling, the number of samples is chosen from [32, 64, 128, 256]. We choose the best sample size using validation node sets.

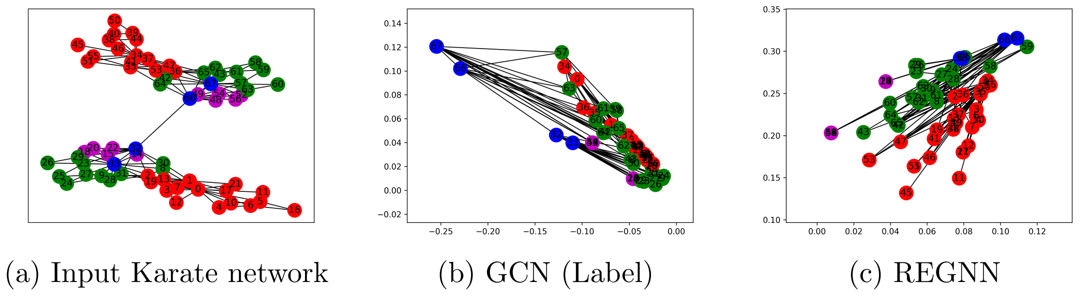

(a) Input Karate network (b) GCN (Label) (c) REGNN

Figure 3.3.: Visualization of node representations from the mirrored Karate network

3.6.4 Results: Synthetic Data

The Karate club network [72] is a graph that is composed of 34 nodes and 78 edges. Each node presents a member, and edges mean interaction among members. To interpret how our REGNN works, we construct a network, which is composed of two copies of the network as in Figure 3.3a. The two networks are connected between node 32 and 66. The colors in the graph were chosen according to community IDs after community detection [73]. Figure 3.3b and Figure 3.3c show the learned representations of nodes from GCN and REGNN, respectively. Similar to GCN [5]'s experiment with the Karate network, a hidden layer with size 2 was inserted before the final softmax layer for the immediate 2-D visualization of latent representations. The representations of the second layer are directly visualized in Figure 3.3a. Labels for training data were chosen from two nodes per each community ID (total 8 nodes.) In the result, while GCN fails to distinguish red and green nodes (i.e., the communities overlap), while REGNN separated the nodes from the two communities better. GCN aggregated nearby labels based on direct edges, so it could not learn from the *role* similarity among nodes in the graph. Meanwhile, our attention layer in REGNN successfully learned and exploited role equivalence by adaptive aggregation.

3.6.5 Results: Real-world Data

Tables 3.3-3.7 show REGNN node classification performance on the Citeseer, Cora, Facebook, PubMed, and Friendster data as proportion of labeled nodes is varied, compared to other baselines. For GCN, Node2Vec, GraRep, Struct2Vec, VERSE, and NEU, we directly obtain results from official implementations. Classification results of all methods are averaged at each proportion. Bold scores represent the corresponding model is significantly better than the others by paired t-tests ($p\text{-value} < 0.05$). In all datasets, REGNN has consistently good results across all label proportions. On the other hand, N-GCN is similar to GCN and LP in Citeseer and Cora, in particular. This indicates that the high-order path information did not help to find better node representations, but the attention over high-order paths helped to increase performance when only known labels are given. Node2Vec and GhostEdge exhibit similar results in most of the datasets, and both achieve good performance at lower label proportions. However, their relative performance often decreases when more labels are available (e.g., in Cora). Struct2Vec and VERSE are not as good as Node2Vec. Since Struct2Vec considers structural similarity only, it does not perform well on most of the datasets. VERSE also learns similarities from Personalized PageRank, which is not helpful for our citation and social network datasets. For PubMed and Friendster, due to the heavy computation cost on the large edges, Struct2Vec, and GhostEdgeNL are not included.

Table 3.8 shows classification performance on the NELL knowledge graph. The result is from the same train/test/validation sets as in [6]. REGNN shows the best performance but is almost on par with Node2Vec and VERSE. However, the execution time for training was much faster than Node2Vec and VERSE. In particular, Node2Vec and NEU incur a great deal of overhead to generate random walks (4,327.43 seconds), and their training time to learn embeddings after the generation was also slower than REGNN. We also tested N-GCN with our importance sampling but the accuracy was still lower than REGNN.

Table 3.3.: Accuracy (%) on Citeseer

% Labeled	10	20	30	40	50
REGNN	**55.03**	**59.05**	**63.18**	**66.84**	**68.74**
N-GCN	51.92	56.84	60.62	64.23	66.07
GCN	51.47	57.40	61.75	65.03	67.36
LP	53.80	57.78	61.37	63.98	66.33
NEU	49.29	55.26	57.54	59.05	59.98
GraRep	50.08	51.97	52.59	52.87	53.48
VERSE	36.28	39.63	40.30	40.66	40.63
Struct2Vec	36.65	39.67	41.97	43.35	43.54
Node2Vec	52.64	54.50	56.05	56.87	57.49
GEdgeNL	50.12	53.94	56.26	58.39	59.49

Table 3.4.: Accuracy (%) on Cora

% Labeled	10	20	30	40	50
REGNN	**76.04**	**80.04**	**82.37**	**84.19**	**85.45**
N-GCN	72.31	78.16	80.91	81.55	84.33
GCN	71.75	77.60	80.93	82.80	84.89
LP	73.55	77.91	80.32	82.81	84.31
NEU	72.28	76.16	79.72	81.55	83.15
GraRep	72.85	74.71	75.02	75.16	75.26
VERSE	57.02	61.53	63.44	63.82	64.32
Struct2Vec	53.81	58.34	61.24	63.38	63.95
Node2Vec	**76.44**	77.88	79.24	80.20	80.04
GEdgeNL	72.22	75.16	77.19	78.79	79.39

Table 3.5.: Accuracy (%) on Facebook

% Labeled	10	20	30	40	50
REGNN	**59.85**	**60.75**	**61.53**	**61.39**	**62.05**
N-GCN	58.27	59.87	60.31	60.49	61.62
GCN	55.72	56.47	59.06	59.17	59.87
LP	56.25	57.36	58.45	59.54	59.83
NEU	56.29	58.52	59.94	60.23	60.88
GraRep	57.48	58.09	59.73	59.50	59.71
VERSE	53.94	56.67	57.09	56.89	57.40
Struct2Vec	53.32	54.47	54.75	54.86	53.56
Node2Vec	57.20	58.07	59.95	59.70	60.36
GEdgeNL	56.28	57.54	58.99	59.61	59.83

Table 3.6.: Accuracy (%) on Pubmed

% Labeled	10	20	40	60	80
REGNN	**79.95**	**82.00**	**83.21**	**83.30**	**84.10**
N-GCN	78.24	81.04	81.41	82.72	83.23
GCN	77.94	80.73	**83.33**	**83.77**	**84.36**
LP	78.97	80.62	82.12	82.75	83.28
NEU	75.59	76.71	77.52	77.94	77.86
GraRep	79.14	79.68	79.90	80.04	80.07
VERSE	**80.44**	81.01	81.15	81.29	81.18
Node2Vec	79.42	80.28	80.86	80.82	81.03

Table 3.7.: Accuracy (%) on Friendster

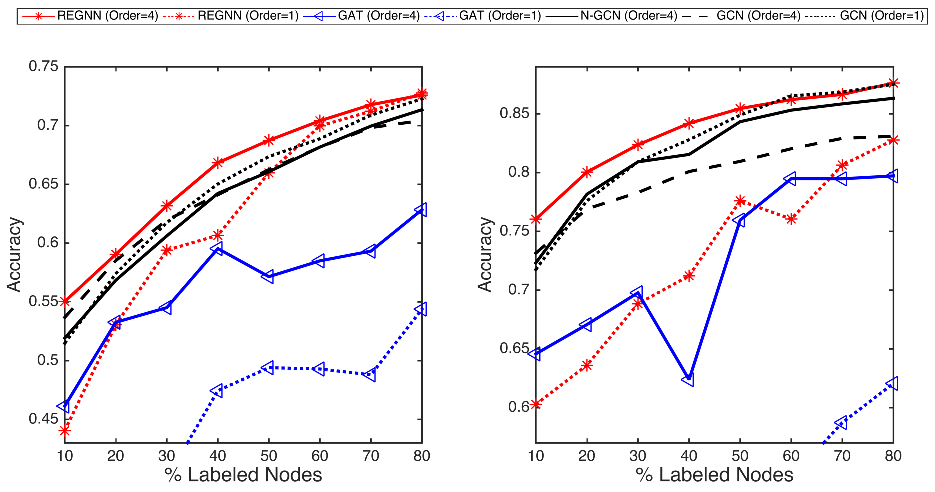
% Labeled	10	15	20	25	30
REGNN	**34.62**	**36.18**	**36.7**	**36.93**	**37.02**
N-GCN	28.93	28.54	32.03	32.55	31.89
GCN	29.7	29.74	30.2	30.18	31.82
LP	27.13	26.32	25.74	24.43	24.43
NEU	30.28	30.75	31.09	31.13	31.4
GraRep	33.53	33.93	34.22	34.53	34.72
VERSE	32.41	33.33	34.01	33.9	34.32
Node2Vec	31.81	32.58	32.8	33.27	33.36

Table 3.8.: Accuracy (%) on NELL

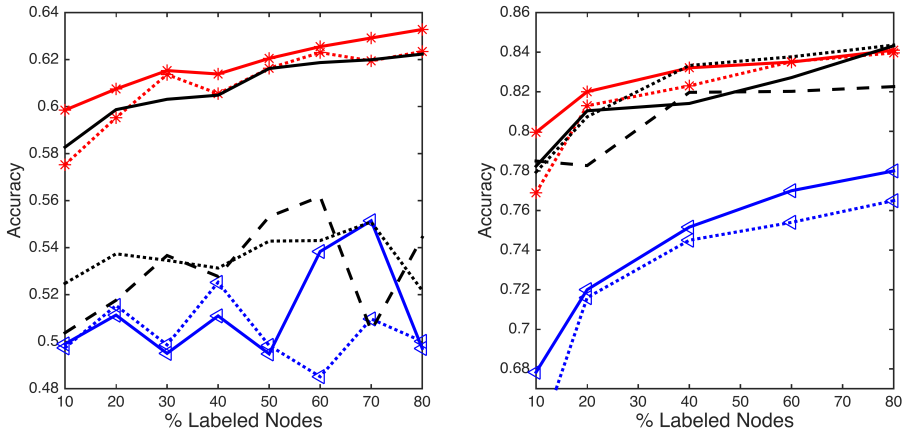
Method	Accuracy	Execution Time (Secs)
REGNN	**85.6**	740.45
N-GCN with IS	84.22	682.23
GCN	79.56	523.27
LP	82.67	1445.41
NEU	81.25	2787.72 (training only)
GraRep	79.25	2339.96
VERSE	**85.43**	1908.54
Node2Vec	84.41	2501.8 (training only)

Effect of Attention Mechanism.

REGNN uses role-based attention to leverage high-order paths. In this section, we report how high-order paths or role-based attention contributes to increasing REGNN's performance. Fig. 3.4 shows comparisons from an ablation study. We compare REGNN (Order=4), which is the best performing order chosen during parameter selection on the validation data, to REGNN (Order=1), which denotes a simplified REGNN that still use the role-based attention but does not consider high-order paths. N-GCN and GAT (Order=4) correspond to versions of our model where the *role equivalence* attention is replaced by the mixing layer used in [74] and the edge-



(a) Citeseer (b) Cora



(c) Facebook (d) PubMed

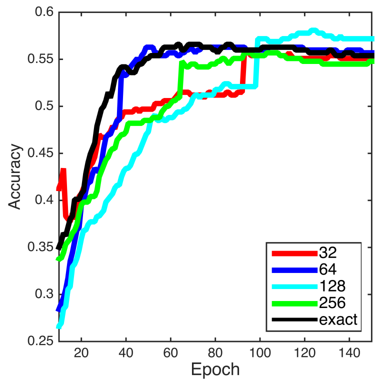
Figure 3.4.: Effect of attention mechanism (Y-axis: Accuracy).

wise attention of [31], respectively. For the mixing layer, column-wise concatenation is used. We also compared with the softmax attention of [74] in our experimental setting, but the concatenation-based mixing layer was more accurate.

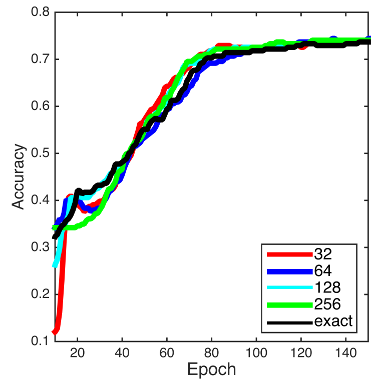
In the ablation experiments, REGNN (Order=4) again achieved the best results across all datasets. Specifically, it performed significantly better (assessed by paired t-tests) than REGNN (Order=1), GAT (Order=4), and GAT (Order=1). In this ablation study, before computing the attentive weights, high-order GCNs are used in the same way for the GAT for fair comparison, but the result is still worse than REGNN (Order=4). We tested different numbers of multi-headed attentions for GAT, but it did not help much. This means that our attention mechanism can identify more meaningful neighbors than the one used in GAT—at least in our application settings, which focus on label propagation in graphs with few attributes. In addition, when high-order GCNs are not used, REGNN (Order=1) is worse than the simple GCN (Order=1) in Citeseer and Cora. This indicates that it is more effective when REGNN combines its latent representations with high-order paths.

3.6.6 Effect of Importance Sampling

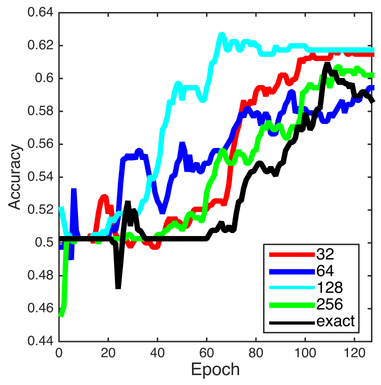
REGNN uses importance sampling to increase scalability. Figure 3.5 shows classification results with different sampling sizes, S. The results were recorded from one of the cross-validation sets, and the ratio of labeled nodes is 10 percent. The curves on the plots are from the accuracy of prediction for validation node sets during training. Each line represents the result of different sampling size, and "exact" means the result of REGNN without importance sampling. In the figure, the learning results with small sampling sizes show good convergence. For example, the experiment of Citeseer shows faster convergence at size 64, compared to "exact", and others converge later. In Cora and PubMed datasets, we cannot distinguish the difference across results from different sampling sizes, and validation accuracies with small sizes converged slightly faster. Validation accuracies on Facebook dataset also show faster convergence when a sample size of 128 is used, and it was even better than "exact". In the full experiment with cross-validation, this was better than results using "exact". This implies that the class labels of individuals in the Facebook dataset could be effectively learned better by sampling structurally similar high-degree nodes. Overall, the results show that importance sampling can achieve similar performance even though it is much more space and time-efficient than the original batch learning.



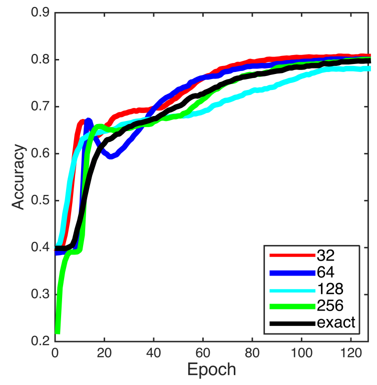
(a) Citeseer



(b) Cora



(c) Facebook



(d) PubMed

Figure 3.5.: Validation accuracy with respect to number of epochs: Each line shows result of different sampling size, S.

3.7 Concluding Remarks

In this chapter, we described REGNN, a neural network architecture that can learn jointly from static and temporal neighborhood structure. The architecture exploits the interactions among local neighbors over time, by learning the temporal evolution of a low-dimensional embedding from a GCN, and models its static neighborhood with a densely connected NN. REGNN is able to improve classification performance by utilizing both patterns in social interactions over time and the set of nodes in the aggregate relational neighborhood.

4 EXPLOITING INTERACTION LINKS WITH GRAPH NEURAL NETWORK FOR NODE CLASSIFICATION

Node classification has been an important problem in relational machine learning. In complex network domains, node classification methods have used different types of relational information such as direct neighbors [75] and autocorrelation of class labels [76]. While these methods have shown the effectiveness of using neighbor information to improve node classification, the majority of real-world network datasets have sparse link structure which limits the amount of neighbor information. Recent work [2, 5, 7, 77–80] on low-dimensional node embeddings and neural network architectures for graphs has shown promising results on node classification for addressing the sparsity problem in static graphs. In particular, GCN [5] learns individual node embeddings by passing, transforming, and aggregating node feature information in an end-to-end fashion and shows promising results on node classification.

However, in real-world domains, network interactions are often changing and evolving over time. For example, users develop their connections to each other in social networks while communicating with others, which could be potentially useful for identifying their class labels (e.g., political view.) Nonetheless, it can be difficult to leverage the dynamics of temporal interactions for node classification. In particular, when the interaction edges are very sparse in each temporal snapshot or a node's neighborhood is biased toward a particular class label in different temporal snapshots, the use of temporal patterns may not be useful for predicting class labels. In these cases, static and aggregated patterns may help to offset these issues. Our proposed model can not only learn temporal interaction patterns, but also model the aggregated neighborhood for node classification.

In this chapter, we first propose a novel deep neural network architecture for sequences of interaction graphs, which is called TSGNet. Our TSGNet leverages the strengths of both the GCN to discover interaction patterns at each temporal snapshot and a recurrent unit, LSTM [81], to capture complex long-term dependencies. For learning the temporal representation more efficiently, a mini-batched training via importance sampling is also proposed. The importance sampling reduces the recursive neighborhood expansion across layers and helps to decrease time complexity while maintaining its performance. Moreover, TSGNet learns a second node representation from a static summary of each node's neighborhood, using a *neighbor encoder*. The static and temporal components of the model are jointly estimated. For evaluation, we conduct extensive experiments on both synthetic and five different types of real-world networks with or without attributes, and we observe significant performance gains compared to state-of-the-art methods. Moreover, a careful ablation study is conducted and shows that our architecture design is most robust across all experimental settings using different static and temporal components.

4.1 Motivation

Figure 4.1 shows examples of interactions in complex networks. Each node and edge indicate an author and a co-author event, respectively. Note that colors represent class labels, yellow for NLP and grey for Database group. In Figure 4.1a, user A and B have the same coauthors (neighbors) when they are aggregated. In this context, the existing graph embedding approaches (i.e., Node2Vec [7] or Graph Convolutional Network (GCN) [5]) will end up learning similar latent representations for the two nodes. However, their temporal interaction patterns could be different. When author A collaborates with other authors differently over time, compared to author B, their representations should be modified accordingly. Meanwhile, if node A and B show similar interaction patterns over time, then it may be difficult to determine the correct class labels through temporal patterns only. In Figure 4.1b, although the temporal coauthoring patterns around author A and B are similar, their neighborhoods on the aggregated graph are entirely different. In that case, using the

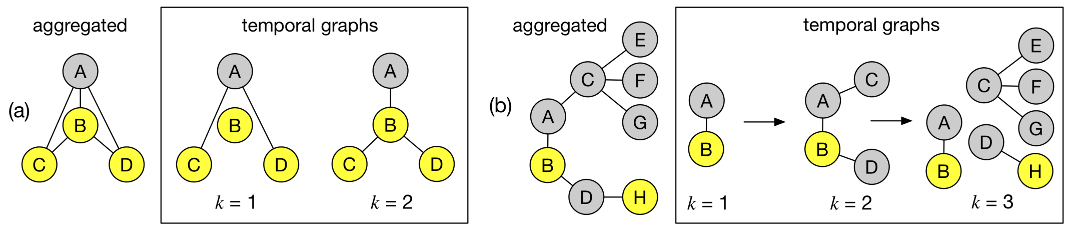


Figure 4.1.: Examples of interactions over time (k): (a) User A and B have same neighbors when aggregated, but different interaction patterns over time (b) User A and B have same interaction patterns over time but different neighborhood patterns when aggregated.

aggregated neighborhoods (which are static features) class labels of node A and B could be identified.

In this work, our goal is to jointly learn patterns in both interactions over time and in static neighbor sets—for node classification. In order to model both properties, we propose a neural network model, TSGNet. The details are described in Section 4.4.

4.2 Notation

We define a graph sequence as a set of graphs such that $G = [G_1, G_2, ..., G_m]$. Each G_k has the same set of nodes, $v_i \in V$ where $\forall i \in [1, n]$, but a different set of edges, $E_k \subseteq V \times V$ such that $G_k = \langle V, E_k \rangle$. If $e_{ij} \in E_k$, there is an edge between v_i and v_j at time k, otherwise there is not. Alternatively, let $A = [\mathbf{A_1}, \mathbf{A_2}, ..., \mathbf{A_m}]$ be the set of adjacency matrices for G, where $\mathbf{A_k}[i, j] = 1$ if $e_{ij} \in E_k$, 0 otherwise.

While the network structure does change over time, we assume that the node attributes do not change over time.[1] Let F be the feature (attribute) set over the nodes. Each $v_i \in V$ has a corresponding feature vector $\mathbf{f}_i \in F$ describing each node. Y is the label set over the nodes. Only a subset of the nodes, $v_i \subseteq V$, have a class label, $\mathbf{y_i} \in \mathbb{R}^{|C|}$, where C is a set of class labels. The goal is to learn a model from the

[1]The setting is realistic for many social and interaction graphs because their attributes available are from basic profiles, resources, or given properties, so they are mostly static. For example, in Facebook, gender and religious views could be attributes, and pre-determined values like contents-rating and budgets could be used for attributes of a social network from IMDB.

partially labeled network and use the model to make predictions $\hat{\mathbf{y}}$ for the unlabeled nodes $\{v_i\} s.t. \mathbf{y_i} \notin Y$. In this work, we assume that Y can be multi-labeled. Moreover, each prediction $\hat{y}_i$ for v_i has an estimated probability.

4.3 Related Work

Supervised Node Classification for Temporal Graph There is some work on relational models that consider temporal patterns with the aim of improving node classification. TVRC [82] attempts to model temporal structures through a two-step process. The key idea behind the TVRC is to model the temporal patterns through an exponential weight decay kernel, where the implicit assumption is that network structure in recent past is more important than the structure in the earlier past. In addition, DDRC [83] proposed a convolutional neural network architecture with max-pooling for node classification, which models temporal interactions among a node's neighbors. DDRC shows stable performance in spite of different variability of neighbor vectors. However, its effectiveness was partially shown in long and relatively denser graph sequences. Our TSGNet is evaluated from more diverse and larger graph datasets and shows better performance in our experiment.

Dynamic Node Embedding Recently, dynamic network embedding approaches were proposed by [84–86] by using spectral updates over time for general relational tasks including node classification. However, they are evaluated in a synthetic setting, in which two temporal snapshots are created by assigning a random timestamp to each edge. Moreover, attributes are also not exploited for learning temporal representations. DynamicTriad [59] also attempted to learn nodes' evolutions through representation learning for general relational learning tasks using multiple temporal graph snapshots, but their effectiveness on node classification is still not clear. For example, they are limited to a specific type of evolution strategies, and attributes are also not used for learning. In our evaluation, the Dynamictriad is compared to our TSGNet.

Model Categorization Table 4.1 categorized TSGNet and other baseline models according to the types of relational information they use (w.r.t. edges and attributes) as inputs to their models. TSGNet uses dynamic interaction edges with or without attributes. In our experiments, we compare TSGNet to all the listed models. Logistic Regression (LR) and Multi-layered Neural Network (NN), Node2Vec [7], and an attributed node embedding, ASNE [87], are also employed to model static edges. See Section 4.5 for more detail. The colors in the table will be used later in the experiments to highlight the performance achieved using each type of relational input.

Table 4.1.: Models categorized w.r.t., the types of relational information they use as inputs.

	No Attr.	Static Node Attr.
Dynamic Edges	**TSGNet**, DynamicTriad	**TSGNet**, DDRC
Static Edges	Node2Vec, LR, NN	GCN, GraphSAGE, ASNE, LR, NN
No Graph	*not applicable*	LR, NN

4.4 TSGNET for Node Classification

Figure 4.2 represents the overall architecture for TSGNet. The TSGNet is composed of (1) a static neighbor encoder and (2) multiple layers of GCN for modeling interaction graphs at each time step k. The details of each are described below.

GCN layers We use GCN as a basic component for modeling each temporal graph, $A = [\mathbf{A_1}, \mathbf{A_2}, ..., \mathbf{A_m}]$. Before the data is fed into the GCN, we use the symmetric normalizing trick described in [5]. $\mathbf{D_k}$ is the diagonal degree matrix of $\mathbf{A_k} + \mathbf{I}$, and $\mathbf{I}$ is an Identity matrix:

$$\check{\mathbf{A}}_{\mathbf{k}} = \mathbf{D_k}^{-1/2}(\mathbf{A_k} + \mathbf{I})\mathbf{D_k}^{-1/2}. \tag{4.1}$$

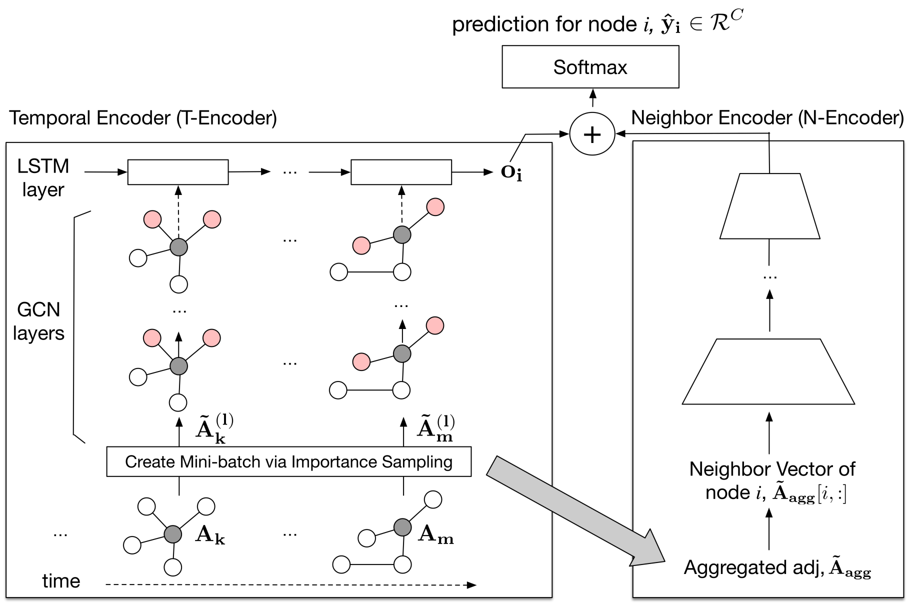

prediction for node i, $\hat{\mathbf{y}}_\mathbf{i} \in \mathcal{R}^C$
Softmax
Temporal Encoder (T-Encoder)
Neighbor Encoder (N-Encoder)
LSTM layer
$\mathbf{o_i}$
GCN layers
$\tilde{\mathbf{A}}^{(\mathbf{l})}_\mathbf{k}$
$\tilde{\mathbf{A}}^{(\mathbf{l})}_\mathbf{m}$
Create Mini-batch via Importance Sampling
$\mathbf{A_k}$
$\mathbf{A_m}$
time
Neighbor Vector of node i, $\tilde{\mathbf{A}}_\mathbf{agg}[i,:]$
Aggregated adj, $\tilde{\mathbf{A}}_\mathbf{agg}$

Figure 4.2.: Architecture for *TSGNet*

Each GCN layer produces node-level output $\mathbf{H}^{(\mathbf{l+1})}_\mathbf{k} \in R^{|V|\times Z^{(l)}}$ where $|V|$ is the number of nodes and $Z^{(l)}$ is the size of output representation per a node, which is determined by $\mathbf{W}^{(\mathbf{l})}_\mathbf{k}$. The outputs are generated at each time step, k.

$$\begin{aligned} \mathbf{H}^{(\mathbf{l+1})}_\mathbf{k} &= f(\mathbf{H}^{(\mathbf{l})}_\mathbf{k}, \breve{\mathbf{A}}_\mathbf{k}) \\ &= (\text{ReLU}(\breve{\mathbf{A}}_\mathbf{k}\mathbf{H}^{(\mathbf{l})}_\mathbf{k})\mathbf{W}^{(\mathbf{l})}_\mathbf{k})) \end{aligned} \tag{4.2}$$

Note that each GCN layer has its own $\mathbf{W}^{(\mathbf{l})}_\mathbf{k}$ at each time stamp. Moreover, GCN [5] originally uses the attribute matrix for $\mathbf{H}^{(\mathbf{1})}_\mathbf{k}$. In this work, we also support a non-attribute option. In this case, the identity matrix is used for $\mathbf{H}^{(\mathbf{1})}_\mathbf{k}$.

Let L be the number of GCN layers for each time step, then the final output for each time step will be an input for an LSTM cell. I.e., $\mathbf{H}^{(\mathbf{L+1})}_\mathbf{k}$ for each time step k, is the temporal input for the k^{th} cell in the LSTM sequence. In Eq. (4.3) below, $\mathbf{o_i}$ returns the final output vector for v_i in the LSTM. The output $\mathbf{o_i}$ is projected a final time using the weight matrix $\mathbf{W}_\text{lstm}$ and bias vector $\mathbf{b}_\text{lstm}$. This will be added to the neighbor encoder for v_i below. If the interaction of v_i ends before the last step, m, it still uses the same $\mathbf{W}_{\text{lstm}}$ and $\mathbf{b}_{\text{lstm}}$ to generate the output $\mathbf{o_i'}$.

$$\begin{aligned}\mathbf{o_i} &= \text{LSTM}(\mathbf{H^{(L+1)}_{1,...,m}}(v_i))\\ \mathbf{o'_i} &= \text{softmax}(\mathbf{W}_{\text{lstm}}\mathbf{o_i} + \mathbf{b}_{\text{lstm}})\end{aligned} \tag{4.3}$$

Neighbor Encoder (NE) The neighbor encoder uses an aggregated matrix $\mathbf{\check{A}_{agg}}$ as input. $\mathbf{\check{A}_{agg}}$ is created by aggregating all elements in $[\mathbf{A_1}, \mathbf{A_2}, ..., \mathbf{A_m}]$ and normalizing in the same way described above. The static component reduces the dimensionality for node v_i from its neighbor vector in $\mathbf{\check{A}_{agg}}$, using stacked fully-connected layers.

$$\begin{aligned}\mathbf{NE_i^{(2)}} &= \text{ReLU}(\mathbf{W^{(1)}}(\mathbf{\check{A}_{agg}}[i,:] \odot \mathbf{h_i}) + \mathbf{b^{(1)}})\\ &= ...\\ \mathbf{NE_i^{(L'+1)}} &= \text{softmax}(\mathbf{W^{(L')}}\mathbf{NE_i^{(L')}} + \mathbf{b^{(L')}}),\end{aligned} \tag{4.4}$$

where $\odot$ means the Hadamard product, $\mathbf{h_i} = \{h_{i,j}\}_{j=1}^{|V|}$. Here, if $\mathbf{\check{A}_{agg}}[i,j] > 0$, $h_{i,j} = \beta$ (for $\beta \geq 1$). Otherwise, $h_{i,j}$ will be equal to 0. This Hadamard product is used to overcome sparsity in the adjacency matrix. If β is 1, it is the same as the adjacency matrix. Otherwise, it puts more weight on non-zero elements in the matrix. It is expected that β will make larger outputs and offset issues from sparsity. In experiments, we set $\beta = 20$, and using this we observe up to 4% of improvement in accuracy.

Addition Layer The element-wise addition layer combines the outputs from the GCN-LSTM and the Neighbor Encoder. Specifically, we compute $\mathbf{\hat{v}_i}$ as the element-wise addition of the outputs from the GCN-LSTM (Eq. 4.3) and the Neighbor Encoder (Eq. 4.4):

$$\mathbf{\hat{v}_i} = \mathbf{NE_i^{(L'+1)}} + \mathbf{o'_i}. \tag{4.5}$$

The addition layer enables a joint representation learned from both the static and temporal neighborhoods around the node. An additional benefit is that the addition layer does not introduce extra parameters, nor does it increase computational complexity. Then, the output $\hat{\mathbf{v}}_\mathbf{i}$ is put though another softmax layer for classification:

$$\hat{\mathbf{y}}_\mathbf{i} = \mathrm{softmax}(\mathbf{W}_{\text{final}}\hat{\mathbf{v}}_\mathbf{i} + \mathbf{b}_{\text{final}}). \tag{4.6}$$

Here, $\hat{\mathbf{y}}_\mathbf{i}$ is the vector output of the softmax function, and each dimension $\hat{y}_{i,j}$ represents the predicted probability of the corresponding class j, given the inputs. For learning, we use categorical cross-entropy (over V_L, the set of labeled nodes) as a loss function at the final layer as:

$$L = -\sum_{i \in V_L} \sum_{j}^{|C|} y_{i,j} log(\hat{y}_{i,j}). \tag{4.7}$$

Since all activation functions are differentiable, learning is simply done via back-propagation.

Importance Sampling When we use multiple GCN layers, the recursive neighborhood expansion across layers poses time and memory challenges for training with large graphs. To overcome this limitation, we leverage the importance sampling again here, which is introduced in Section 3.4.3 in Chapter 3. In short, at every epoch, all nodes are randomly divided to create a mini-batch set, B, which is composed of multiples of γ nodes. We set $\gamma = 1024$. B provides a candidate node set for sampling $|S|$ later. When it comes to a new *mini-batch*, $\tilde{\mathbf{A}}_\mathbf{k}$ is induced from $\check{\mathbf{A}}_\mathbf{k}$ according to the S. Similarity, the input matrix of neighbor encoder, $\check{\mathbf{A}}_\mathbf{agg}$, is replaced by $\tilde{\mathbf{A}}_\mathbf{agg}$. The inner for-loop for $\check{\mathbf{A}}_\mathbf{agg}$ retains only the edges in S, but maintains the dimensionality of the NE.

Complexity Analysis Recall that m is the number of temporal GCNs and $|F|$ is the number of node attributes. Then assuming the number of hidden units in LSTM, GCN, and NE is constant, the computational complexity of TSGNet is $O(|V| + m \cdot |F| \cdot |E|)$, where $O(|V|)$ is from neighbor encoder and $O(m \cdot |F| \cdot |E|)$ is from the set of temporal GCNs. Because m and $|F|$ are typically much smaller than $|E|$, the time complexity is linear in the number of edges, i.e., $O(|E|)$. With importance sampling, the complexity becomes $O(|V| + |E_S|)$, where $|E_S|$ is the number of edges induced in S. When the non-attribute option is chosen, the complexity is still $O(|V| + |E_S|)$ because the input identity matrix is sparse, thus $|F| = 1$ in the sparse representation, and can be considered as a constant with sparse-dense matrix multiplication.

4.5 Experimental Results

4.5.1 Data

We use three real-world network datasets for evaluation. Table 4.2 reports brief statistics for each network.

Table 4.2.: Network data statistics. m is the number of time windows, and other notations are from Section 4.2.

	$\|V\|$	$\|E\|$	m	$\|F\|$	$\|C\|$
Facebook	2,716	22,712	55	2	2
DBLP	17,191	318,735	18	2,997	2
IMDB_G	5,043	43,494	65	73	2
IMDB_R	92,611	472,630	14	-	2

Facebook The Facebook network was scraped from a University group [9]. Each user (node) is associated with political views for their class labels. An edge is formed when a user writes a post to his or her friend's wall. Users who posted more than 1 times a week during at least 7 weeks are chosen. A time window is defined with 2 weeks. Attributes for Facebook are religious views and gender.

DBLP For this dataset, a co-authorship network is extracted from DBLP, and an edge is created when two authors published one paper together at the time step. Thus, nodes represent authors. Publication venues are selected from AI/NLP and

DB conferences[2]. Authors are selected when they have at least 7 years of publication history, and the most published area is chosen for the class label of each author. For attributes, term vectors from titles of each user's published papers were used.

IMDB_G (Gross Income) We use Kaggle's IMDB (Internet Movie Database) 5,000 movie dataset. An edge is formed when two movies share an actor or actress at each year. All movies have a least 2 temporal edges. A movie has a positive label if the movie gross is larger than 10 million dollars. For this work, we choose budgets, content rating, the number of faces in a movie poster, and genres as features. The budgets are quantized from 0 to 9 using percentiles. Each feature is transformed into one-hot encoding representation.

IMDB_R (Rating) This dataset is from the whole IMDB database[3], and all participants including actors and writers are imported. An edge is formed when two movies share any crew each year. When a movie's rating is larger than 7.0, it is chosen as a positive label. The periods of all movies are from 2005 and 2018. There are many missing values to consider all movies, so attributes are ignored in this dataset. All movies have at least 11 temporal edges.

4.5.2 Comparison Models

Logistic Regression (LR) Logistic regression is performed using neighbor vectors with L1 regularization. This allows us to compare how relational and temporal patterns improve performance. The aggregated (binary or degree normalized (weighted)) graph of all temporal graphs is used for training.

[2]AI/NLP: IJCAI, AAAI, SIGIR, ECIR, CLEF, CHIIR, AIRS, ACL, EMNLP, and COLING; DB: ICDE, VLDB, SIGMOD/PODS, and EDBT from 2000 to 2017.
[3]The IMDB dataset was downloaded in November 2018

LSTM We use the TSGNet's input representation for the LSTM, but the GCN layers and the *neighbor encoder* are not used in the architecture. For inputs, the first-hop neighbors at each time window are fed directly into the LSTM layer.

GCN and GraphSAGE To compare TSGNet with Graph neural networks, GCN [5] and GraphSAGE [10] are evaluated with the aggregated (binary) static graph input. Attributes are used in the same way with TSGNet. We used an LSTM aggregator for GraphSAGE. Because other aggregators for the GraphSAGE such as GCN, mean, and pool are worse than LSTM, the results are not reported here.

Node2Vec For learning a static node embedding method, Node2Vec, we set $d = [16, 32, 64]$, $r = 10$, $l = 80$, $k = 10$, and p and q were searched over [0.5, 1, 2]. The aggregated (binary) matrix is used for its training.

ASNE ASNE [87] is a recent attributed node embedding method. We used the same hyper-parameter search criteria as in [87].

DDRC DDRC [83] is a CNN-baed temporal classifier, which considers interactions over time. This does not have a *neighbor encoder* or GCN component. The inputs are used as in LSTM above.

Multi-layered Neural Network (NN) For NN, the *neighbor encoder* of TSGNet is used for training and testing.

TempGCN (GCN+LSTM) This is a version of the TSGNet without the *neighbor encoder (NE)* component where we use GCN to model the temporal graphs with an LSTM.

DynamicTriad A dynamic node embedding method, DynamicTriad [59], is also tested with all combinations of parameters as in [59]. Both unweighted graphs and weighted graphs were used for learning, and, edges are weighted by the number of common neighbors during each period.

4.5.3 Evaluation Methodology

Every result we report is from the average of 10 trials using randomly shuffled node sets. Note that the entire graph is known before learning, and 70%, 20%, and 10% of node labels are used for training, testing, and validation, respectively. If the accuracy on the validation dataset does not increase during 5 epochs, learning stops. We also use dropout regularization (0.2) and rectified linear units for activation functions. For optimization, we use the *adam* optimizer [88] to update variables. For TSGNet, LSTM, and GCN, the number of hidden nodes is searched over [8, 16, 32, 64], and the numbers of hidden nodes in the neighbor encoder are 512 and 128 at each layer. In addition, there are three GCN layers for TSGNet. For importance sampling, the sampling size $|S|$ is chosen from [8, 16, 32, 128, 256].

4.5.4 Results: Synthetic Data

To evaluate the concept of TSGNet, we generate synthetic data from two simplified-Dynamic Stochastic Block Models (DSBM) [89] to evaluate our model. We set the number of users to 100, and the length of time-windows for each node is determined by $25 + \text{uniform}(0, 1) \times 25$. The first DSBM (Dense Block Model) is composed of 4 different partitions, P_1, .., P_4, at time$(k)\%2 == 1$. Each partition is composed of 50$\times$50 nodes. In all other time-windows, edges are generated from 9 partitions, P'_1, .., P'_9. Each partition has different edge probabilities, as in Figure 4.3. The second model (Sparse Block Model) is designed to generate sparse DSBM with low probability. All other conditions are the same, but each probability is 10 times sparser than the dense block model. For class labels, 0 is assigned for the first half of nodes (thus, senders of P_1 and P_2), and 1 is set to the second half.

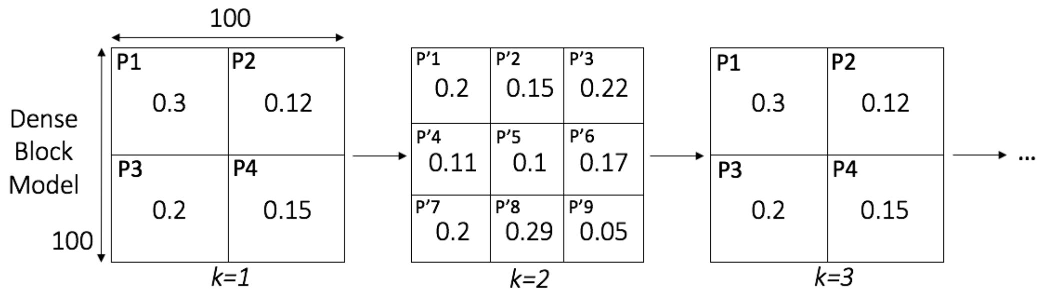


Figure 4.3.: Dense Block Model used to generate synthetic networks. Sparse Block model is 10 times sparser than the Dense Block Model.

Node classification accuracy (and standard errors) on the synthetic data are shown in Table 4.3. Bold numbers indicate statistically significant top results. For data from both sparse and dense block models, our proposed model exhibited good performance. While DynamicTriad, DDRC, and LSTM show better performance than TSGNet in the dense data, they are worse in the sparse block model. Meanwhile, other classifiers (GCN, LR, and NN), which were originally designed for static graphs, showed worse performance than TSGNet's results. Node2Vec works well for the sparse data, but it is worse than TSGNet (p-value < 0.05 in paired t-test).

Table 4.3.: Classification accuracy on synthetic data. Values in ( ) denote standard errors. Colors indicate type of relational input used by the model from Table 4.1.

	Dense Block Model	Sparse Block Model
TSGNet	0.89 (0.0299)	**0.884 (0.0282)**
DynamicTriad	**1.0 (0.0)**	0.615 (0.0316)
DDRC	**1.0 (0.0)**	0.705 (0.026)
LSTM	**1.0 (0.0)**	0.530 (0.0376)
GCN	0.519 (0.0272)	0.504 (0.0277)
Node2Vec	0.494 (0.036)	0.771 (0.026)
LR	0.429 (0.0164)	0.63 (0.227)
NN	0.485 (0.0212)	0.688 (0.0326)

4.5.5 Results: Real-world Data

Performance Without Node Attributes Table 4.4 shows the classification results for the three different real-world datasets. Note that bolded numbers indicate statistically significant top results. (Weighted) in LR refers to versions where the input matrices of the corresponding methods are normalized by the number of edges per each node. In the experiment, TSGNet exhibited the best performance over other alternatives for all datasets and shows comparable performance to TSGNet w/o IS (Importance Sampling). While simple static classifiers such as LR and NN return good performance for Facebook (FB) and DBLP due to the high correlation between neighbor vectors and class labels, however, they are still worse than our TSGNet. These characteristics make TempGCN more difficult to model the data because it is too complex to learn the simple neighborhood. Despite that, the neighbor encoder component of the TSGNet helps it learn the hidden dependencies among nodes and their static neighborhood well. As a result, it produces a significant gain in performance. DDRC and LSTM showed poor performance because the data is also very sparse. DynamicTriads are better than GCN and GraphSAGE in IMDB_G but still worse than TempGCN and TSGNet. Overall, TSGNet produces an average reduction in classification error of 16%, compared to GraphSAGE, which is the best competitor.

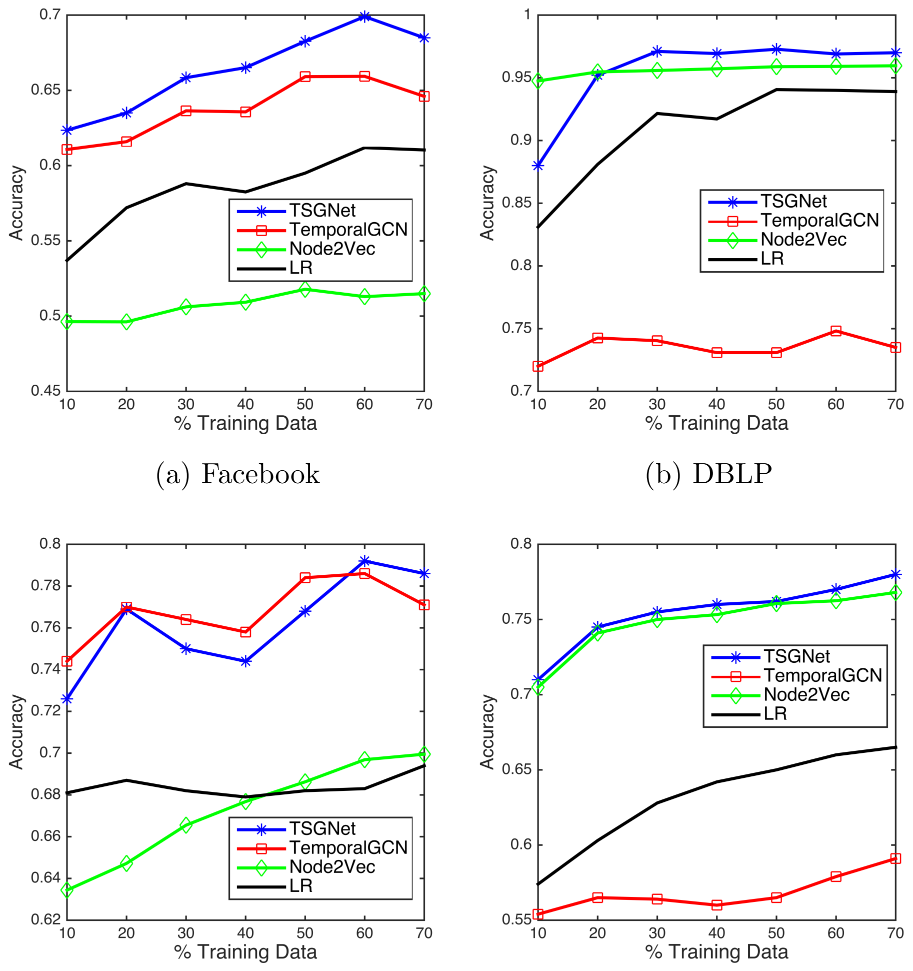
Figure 4.4 shows learning curves on the three datasets as we vary the amount of training data. The learning curves compare the performance as the number of training nodes increases. Note that the set of nodes for testing and validation is same across all range of x-axis. Although the number of training nodes was controlled to calculate the supervised loss, the complete adjacency matrices at each time step for the GCN layers were fixed for the experiment. The experimental assumption was also applied to all other alternatives, TempGCN, Node2Vec, and LR. For Node2Vec, the compete network structure is known for learning representation, and the number of nodes is controlled when its supervised classifier is trained. Therefore, all results with the small training data were not poor. In the Facebook and DBLP datasets, TSGNet

Table 4.4.: Classification accuracy on real-world datasets. Colors indicate relational input type from Table 4.1. Results of DDRC and LSTM for IMDB_R are ignored due to the learning time limit ($\geq$1 day). Bolded numbers indicate statistically significant top results from the paired t-test (p-value<0.05).

	FB	DBLP	IMDB_G	IMDB_R
TSGNet	**0.68**	**0.97**	**0.78**	**0.78**
TSGNet w/o IS	**0.688**	**0.97**	**0.786**	0.771
TempGCN	0.646	0.734	0.77	0.591
DynamicTriad_W	0.542	0.652	0.732	0.657
DynamicTriad	0.534	0.633	0.730	0.645
DDRC	0.554	0.542	0.717	-
LSTM	0.514	0.538	0.696	-
GraphSAGE	0.645	0.963	0.712	0.752
GCN	0.521	0.665	0.719	0.568
Node2Vec	0.515	0.96	0.7	0.768
NN	0.623	0.83	0.716	0.726
LR	0.593	0.939	0.699	0.665
LR_W	0.613	0.955	0.689	0.673

was consistently better than the others. For IMDB_G dataset, TSGNet improved in performance as the size of training set increased.

Performance With Node Attributes Table 4.5 shows classification results when node attributes are incorporated into the models. The result for TSGNet with attributes was better than the other alternatives which used attributes in their input. Moreover, the performance of TSGNet without attributes was even better than the result of the best model which uses attributes. Note that, for the DDRC without attributes, an identity matrix is concatenated to the input neighbor vector. This result indicates that it can learn a good representation with the only structural interactions. (attr. + neighbor) means the concatenated input between attribute and neighborhood vectors. ASNE, LR, and NN with the new input show good results in general, but they are worse than TSGNet. ASNE was bad for DBLP because it could not utilize labels to learn the embedding. Also, GCN did not work well both with and without attributes. GCN is based on a 1-layer perceptron, which is not a universal



(a) Facebook (b) DBLP

(c) IMDB_G (d) IMDB_R

Figure 4.4.: Learning curves for each dataset as amount of training data is varied.

approximator [90]. The 1-layer perceptron in the GCN works like a linear mapping, so the layers may degenerate into simply summing over neighborhood features [28]. With this reason, GraphSAGE with LSTM aggregator can model interaction better than GCN for Facebook and DBLP. Overall, TSGNet with or w/o attributes reduces classification error up to 24% and produces an average reduction in classification error of 10%, compared to GraphSAGE.

Table 4.5.: Classification accuracy on real-world datasets with node attributes. Colors indicate relational input type from Table 4.1. Bolded numbers indicate statistically significant top results from the paired t-test (p-value<0.05).

	FB	DBLP	IMDB_G
With Static Attributes			
TSGNet	**0.675**	**0.96**	**0.777**
DDRC	0.554	0.938	0.749
GraphSAGE	0.655	**0.967**	0.717
GCN	0.483	0.881	0.720
ASNE	0.525	0.601	0.734
LR (attr. + neighbor)	0.664	0.96	0.744
NN (attr. + neighbor)	0.645	0.955	0.759
LR (attr. only)	0.63	0.891	0.756
NN (attr. only)	0.63	0.886	0.735

Temporal Sequence Randomization: Impact on Performance To see the effect of temporal sequence's randomization, the time-windows were randomly shuffled and used for training. The order of words in language models for NLP and speech recognition is quite important to represent sentences, but the temporal order of social interactions could be reversed and often spontaneously happen. In order words, the randomized temporal sequences are likely to represent another instance of evolution. As in Table 4.6, TSGNet and TempGCN also worked well in the randomized inputs and not significantly different from the results of original inputs. These results are also related to Janossy pooling [91], which can make the LSTM a permutation-invariant function. This explains why the randomized inputs often work better with LSTM when there is general dependency among input sequences. For example, collective classification using LSTM [11] shows their effectiveness from the randomized sequences.

Ablation Study of Model Components TSGNet uses GCNs for learning temporal interactions and a NN neighbor encoder for learning the aggregated static first-neighbors. However, we could have chosen other architectures for either component. Note that we did not use importance sampling to see the true effect of each compo-

Table 4.6.: Classification accuracy with different temporal inputs. Colors indicate relational input type from Table 4.1.

	Facebook	DBLP	IMDB_G	IMDB_R
TSGNet	**0.688**	**0.97**	**0.786**	**0.78**
TSGNet (RandOrder)	**0.679**	**0.96**	**0.774**	0.772
TempGCN	0.646	0.734	**0.771**	0.591
TempGCN (RandOrder)	0.658	0.735	0.750	0.573
DDRC	0.554	0.542	0.717	-
DDRC (RandOrder)	0.573	0.54	0.718	-
LSTM	0.514	0.538	0.696	-
LSTM (RandOrder)	0.480	0.53	0.693	-

Table 4.7.: Effect of different joint learning approaches: N-Encoder (N-En) and T-Encoder (T-En) refer to *neighbor encoder* and *temporal encoder*, respectively. Bolded numbers indicate statistically significant top results from the paired t-test (p-value<0.05).

N-En	T-En	FB	DBLP	IMDB_G	IMDB_R
NN	GCN	**0.688**	**0.97**	**0.786**	**0.771**
NN	NN	**0.676**	0.953	**0.776**	0.711
GCN	GCN	0.672	0.652	**0.788**	0.707
–	GCN	0.646	0.734	0.771	0.591
–	NN	0.647	0.732	0.769	0.707
GCN	–	0.521	0.665	0.719	0.658
NN	–	0.623	0.83	0.716	0.726

nent. Table 4.7 shows the results for different variants of the architecture, with the original components of TSGNet in the first row. Instead of the GCN in TSGNet, when we use regular densely-connected NN, its performances decrease in DBLP, as shown in the second row of the table. When the GCN is missing in TSGNet like the last row of the table, it also does not work well. Similarly, when the NN in TSGNet is replaced with GCN layers or an empty layer, we can observe a significant drop in Facebook and DBLP. This represents that our NN-based neighbor encoder helps to jointly learn the temporal network's interaction well if we use GCN layers. In summary, we could see that our TSGNet is never worse than other alternatives.

Parameter Sensitivity We note that the parametrization of the model depends on m, the number of time windows in the data. Figure 4.5 shows how performance varies as we vary how many recent time-windows are used for training. We observe the accuracy of TSGNet seems to plateau after 40 percent of time-windows are used. We also note that performance does degrade as more time windows are used, which indicates that the model is not overfitting to the use of additional temporal interactions.

We also evaluated the sensitivity of TSGNet to β. We searched over values [1-30] for each dataset. For IMDB_G, the accuracy varied from 0.78 to 0.81, and the accuracies of IMDB_R are from 0.761 to 0.78.

Facebook's accuracy also changed from 0.68 to 0.705 depending on the β. For DBLP's the accuracy did not vary much. This indicates that model performance is relatively insensitive to the choice of β.

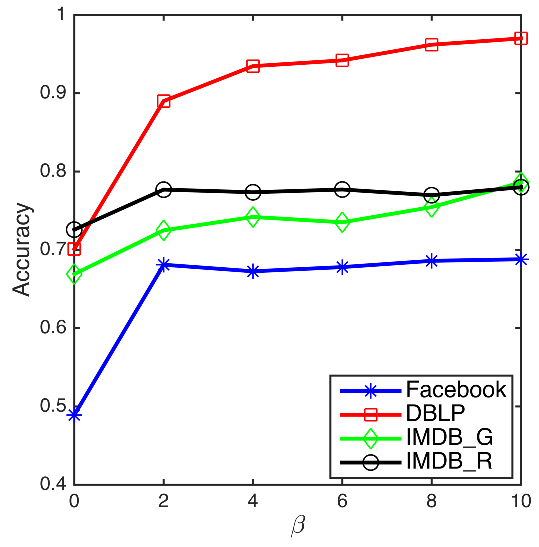


Figure 4.5.: Classification accuracy as m is varied.

4.6 Concluding Remarks

While label propagation has been widely used to improve the performance for semi-supervised learning, existing deep neural networks have not greatly explored the property. In this chapter, we described TSGNet, a graph neural network architecture that exploits label propagation in a message-passing framework. TSGNet not only learns from high-order path information but also adaptively aggregates the hidden representation of neighboring nodes using role equivalence. In our experimental results, TSGNet consistently showed improved performance for semi-supervised node classification. We assess the significance of the gains using paired t-tests and determine that TSGNet significantly outperforms the best alternative (N-GCN) across all label proportions on five real-world network datasets.

5 GENERATING EXPLANATIONS FOR NODE REPRESENTATION LEARNING BY IDENTIFYING IMPORTANT NODES WITH *BRIDGENESS*

Node Representation learning methods (or Node embedding methods) have aroused considerable research interests in recent years. The fundamental problem is how to find low-dimensional representations while preserving the inherent relational properties in an embedding space. (i.e., if neighbors of two vertexes are similar in the original network, they should have similar embedding vectors.) Prior works have demonstrated that, besides the pairwise edges, high-order proximities or structural similarities between nodes are of importance in capturing the underlying structure of the network [1–4] and thus can provide valuable information for learning the embedding vectors.

While the node embedding models have increased the performance of many relational tasks such as node classification and link predictions, it compromises their ease of interpretation. Providing interpretable explanations is a crucial criterion when the embedding models are applied in time-critical and cost-intensive areas such as biology [12] and medicine [13]. Recently, similar research questions have been raised by researchers, and many explanation methods have been proposed to provide explanations with salient features [14, 15] or make the models more transparent [16]. However, most of them have focused on providing explanations for predictive learning models on simple sequenced or grid inputs.

Moreover, since node embedding models such as DeepWalk [1], LINE [2], Struc2Vec [3], and PTE [4] are typically trained in an unsupervised and transductive fashion, the previous explanation methods are not directly applicable to them. Recently, while GNNExplainer [17] is proposed to provide explanations for Graph Neural Network models (e.g., GCN [5]) w.r.t. a single prediction, and it is still required to have an end-task with additional heavy learning. Moreover, it has not been evaluated with real-world graphs. In this chapter, we propose a novel explanation method for finding globally important nodes to the embeddings, which are learned from the aforementioned models. Our analysis and proofs reveal that node-level importance in DeepWalk could be measured by *bridgeness* under cluster-aware local perturbation. The *bridgeness* of a node is usually defined by considering how the node is connected to different clusters, and one example of high *bridgeness* nodes is people in management positions who initiate collaborations between different departments such as software development and marketing. You can refer to its formal definition in Def. 6. We also provide an efficient gradient-based method, which we call GRAPH-wGD, to approximate the *bridgeness*-based ranking. We evaluate our method with five different real-world graphs, and the importance ranks found by GRAPH-wGD show high rank correlations to *bridgeness* in all datasets. Moreover, GRAPH-wGD shows consistently better results than recent alternatives w.r.t. prediction change and new neighbors' emergence after perturbation of the discovered importance nodes and relearning the embedding.

5.1 Related Work

Previous methods mainly aim to provide explanations to the ML models who take non-graph inputs. To understand outputs from graph-based ML models, Degree and PageRank can provide useful information to many downstream tasks. For example, high degree nodes are often regarded as popular nodes [92] in Twitter, and nodes who have high PageRank scores [93] are still highly ranked in modern search engines or recommendation engines. However, they are not post-hoc interpretation methods, and, depending on ML models, the theoretical relationships between the models and the properties are not clear. In particular, although random walk-based node embeddings like DeepWalk heavily rely on high-degree nodes, high-degree nodes are not guaranteed to have the highest global impact on the learned model. For example, in the worst case, if the high degree nodes are placed in a single cluster, and the perturbation on the nodes may not change label prediction or clustering much in the embedding space.

Recently, GNNExplainer [17] has been proposed to provide explanations for graph neural network models (GNNs) with respect to the prediction of a single node. However, the method does not have the capability of providing global explanations for the node embedding itself. Taxonomy induction method [37] and XGNN [56] are also relevant to this work, but they interpret node embeddings only by identifying clusters in the embedding space or providing a model-level interpretation with a graph generation, respectively. Other methods for ML models which take non-graph inputs are discussed in the Section 2.2.2 of Chapter 2.

5.2 Importance and the Impact of High Bridgeness in Node Embeddings

5.2.1 Node Importance

To measure the importance of nodes in embeddings, we generalize the well-known pairwise distance [94,95] to measure the impact on the learned node embeddings after perturbing a graph. Let A be the adjacency matrix of an undirected and weighted graph G, and $d_i = \sum_j A_{ij}$ be the degree of v_i. V is a set of vertices in G. We assume that a learned embedding W takes $v_i \in V$ to return low-dimensional vector $\vec{w_i} \in \mathcal{R}^k$. We first define the importance of each node in the learned embedding using Degree-Normalized Pairwise Distance (DNPD):

Definition 4. *(Node importance in latent space) Let G be an input graph with vertices V and let G' be a perturbed version of G. Let $W \in \mathcal{R}^{|V|\times k}$ and $W' \in \mathcal{R}^{|V|\times k}$ be latent representations learned from G and G', respectively. Then the **importance** of v_b in the latent space is:*

$$\begin{aligned} \textit{Importance}\,(v_b|W,W') &= ||DNPD(W) - DNPD(W')|| \\ &= \left|\left| \sum_{v_i,v_j\in V} \left|\left| \frac{\vec{w_i}}{\sqrt{d_i}} - \frac{\vec{w_j}}{\sqrt{d_j}} \right|\right|^2 - \sum_{v_i,v_j\in V} \left|\left| \frac{\vec{w}'_i}{\sqrt{d_i}} - \frac{\vec{w}'_j}{\sqrt{d_j}} \right|\right|^2 \right|\right|. \end{aligned}$$

Definition 4 provides a conceptual node importance measure for v_b in any latent representation under perturbation. In the next section, we propose a specific perturbation strategy to explore properties of node importance on both spectral and DeepWalk embeddings.

5.2.2 Cluster-Aware Perturbation for Spectral Embeddings

First, we define a degree/volume-preserving cluster-aware perturbation method. We note that the initial set of clusters is given before perturbation. We use spectral clustering to get the set of clusters, $\{C_1, ..., C_k\}$, for ease of theoretical analysis (described next). There are several objective functions that capture the clusters, and we focus on finding eigenvectors to maximize the relaxed normalized association $N_{asso}(C_i|G) = \frac{assoc(C_i, C_i|G)}{assoc(C_i, V|G)}$ [96].

Definition 5. *(Degree/Volume-Preserving Cluster-Aware Perturbation) Given a graph G and its set of clusters $C = \{C_1, ..., C_k\}$, we define a function* Perturb*(G, v_b) = G'_{-v_b} to perturb inter-cluster edges around v_b as follows:*

1. *Set $V_{cand} = \{v_i | v_i \in C_i, A[b, i] > 0\}$*
2. *Sample Q, where is composed of $\alpha \cdot |V_{cand}|$ number of vertices, from V_{cand}*
3. *For $\forall v_q \in Q$, // Remove inter-cluster edge $e(v_b, v_q)$ and $e(v_q, v_b)$, adjust by adding self-edges*
   $A[q, q] = A[q, q] + A[b, q],\ A[b, b] = A[b, b] + A[b, q]$
   $A[b, q] = 0,\ A[q, b] = 0$
4. *Return $G'_{-v_b} = A$.*

Using the Perturb function, we can analyze how $N_{asso}(C_i|G)$ will change after the perturbation.

Lemma 1. *Given a graph G, its set of clusters $C = \{C_1, ..., C_k\}$, and an arbitrary node $v_b \in V$, let G'_{-v_b} = Perturb(G, v_b) from Definition 5. If $v_b \in V$ has at least one inter-cluster edge, then $N_{asso}(C_i|G) \leq N_{asso}(C_i|G'_{-v_b})$ for $\forall C_i \subset C$.*

See appendix for all proofs. Now we introduce the definition of *Bridge Node* [97–99]. In agreement with earlier findings in [100–102], we assume that bridge nodes have more inter-modular positions than community cores. The existence of bridge nodes often leads to more inter-cluster edges. The "bridge" connects the clusters and can enhance the integration of the whole network, and its formal definition is:

Definition 6. *(Bridge Node and Bridgeness) [99] Given $C = \{C_1, ..., C_k\}$ in G, when a node $v_b \in C_b$ connects at least two different clusters including C_b, it is called a Bridge Node. The Bridgeness of v_b is:*

$$Bridgeness(v_b \in C_b|C, G) = \gamma(C_b) \cdot d_b^{inter}, \tag{5.1}$$

where d_b^{inter} is the inter-cluster degree of node $v_b \in C_b$, which is computed from G and C. $\gamma(C_b)$ represents the importance of the clusters C_b. For $\gamma(C_b)$, the number of its neighboring clusters to C_b is used in [99], and the distance to each cluster have also used [98, 103]. In this chapter, we set $\gamma(C_b) = 1$ for simplicity.

The following Lemma shows that the difference in N_{asso} is maximized after perturbing the node with the largest *bridgeness.*

Lemma 2. *Let V be the nodes in graph G with clusters C, and G'_{-v_j} = Perturb(G, v_j) using Def. 5 with the perturbation ratio $\alpha = 1.0$ If we define*

$$v_b = arg \max_{v_j \in V} (N_{asso}(C_1, ..., C_k|G'_{-v_j}) - N_{asso}(C_1, ..., C_k|G)),$$

then v_b is the node with largest Bridgeness.

Now we can derive which node has the largest *Importance* in a spectral embedding (eigenvector U from $D^{-1/2}AD^{-1/2}$) under perturbation as:

Theorem 1. *Let V be a node set in G, and we have k clusters $\{C_1, ..., C_k\}$ in G. G'_{-v_j} = Perturb(G, v_j) using Def. 5 with the perturb_ratio $\alpha = 1.0$, and eigenvectors $U \in \mathcal{R}^{|V| \times k}$ and $U'_{-v_j} \in \mathcal{R}^{|V| \times k}$ are given from G and G'_{-v_j}. Then, we have*

$$arg \max_{v_j \in V} Importance(v_j | U, U'_{-v_j}) = arg \max_{v_j \in V} Bridgeness(v_j | C, G).$$

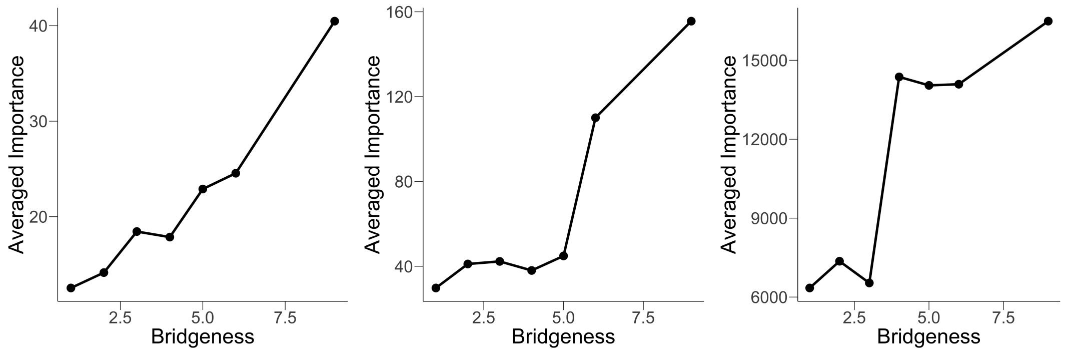
5.2.3 Connection to Node Embedding Models

Similarly, here we consider DeepWalk embedding to see which node has the largest *Importance* under perturbation. We denote $D_{ii} = \sum_j A_{ij}$ as the diagonal degree matrix, T is the window size, b is the number of negative samples, and $vol(A) = \sum_{v_i, v_j \in V} A_{ij}$ is the volume.

Theorem 2. *Let V be a node set in G, and there are k clusters $\{C_1, ..., C_k\}$. G'_{-v_j} = Perturb(G, v_j) using Def. 5 with $\alpha = 1.0$, and eigenvectors $\hat{U} \in \mathcal{R}^{|V| \times k}$ and $\hat{U}'_{-v_j} \in \mathcal{R}^{|V| \times k}$ from DeepWalk embeddings are given from G and G'_{-v_j}. Then,*

$$arg \max_{v_j \in V} Importance(v_j | \hat{U}, \hat{U}'_{-v_j}) = arg \max_{v_j \in V} Bridgeness(v_j | C, G).$$

Other embedding models such as LINE, PTE, and Struc2Vec could be analyzed in the same way. For LINE and PTE, similar theoretical approximations were studied in [78]. As in Table 1 in [78], we can expect the same result under the same perturbation in both LINE [2] and PTE [4] because our perturbation method does not change degree matrices and other constants. Struc2Vec [3] learns low-dimensional representations using Skip-Gram architecture on the context graph, which is constructed by structural similarity among nodes. When we assume input adjacency is the context graph, its bridge nodes are still important as in DeepWalk.



(a) Eigenvector (b) (Approx.) DeepWalk (c) DeepWalk

Figure 5.1.: Averaged importance over bridgeness from Eigenvectors, (Approx.) DeepWalk, and DeepWalk

5.2.4 Verifying Theorems using LFR Benchmark Networks

LFR benchmark [104] is an algorithm that generates realistic benchmark networks, which has been used for evaluating community (or cluster) detection methods. The advantage of the benchmark is that it accounts for the heterogeneity in the distributions of node degrees using a power-law distribution $p(x) \propto x^{-\tau_1}$, and the size of the communities are also taken from the distribution with an exponent τ_2. There is a mixing parameter μ, which is the average fraction of neighboring nodes of a node that do not belong to any community that the benchmark node belongs to. We set the parameters to $|V| = 500$, the minimum number of nodes in each cluster $min(C_k) = 20$, $\tau_1 = 2.5$, $\tau_2 = 1.5$. Figure 5.1 shows the result of averaged importance after perturbing nodes that have different *bridgeness* using Definition 4. The scores are computed on the different latent spaces: (a) Eigenvector of G, (b) Eigenvector of M in Theorem 2 ((Approximated) DeepWalk), and (c) DeepWalk. As we derived in Theorems 1-2, the importance scores of nodes increase when they have more *bridgeness*.

5.3 Proposed Gradient-Based Method to Find Bridge Nodes

In the previous section, we explored that bridge nodes play important roles in terms of the change of node importance in the DeepWalk embedding. However, existing methods to find bridge nodes are $O(|E| + |V|^{2.367})$ due to inter-community degree computation ($O(|E|)$) after spectral graph clustering ($O(|V|^{2.367})$). In particular, spectral clustering relies on eigenvector decomposition (EVD), and to our knowledge, the time complexity of the fastest EVD solver is $O(|V|^{2.367})$ or $O(nnz(A))$ [105]. $O(nnz(A)) = O(A^2)$ if A is dense, where $nnz(A)$ denotes the number of non-zero entries in A.

To mitigate this burden, we propose a gradient-based method to find the bridge nodes. First, we derive a measure by exploiting the magnitude of the gradient, which is also called *Sensitivity* [54]. We note that the magnitude is measured after learning the corresponding embedding model[1]. For example, in order to learn the Skip-Gram architecture, hierarchical softmax takes a pair of nodes, a target (input) node and one of the context nodes, for updating the embedding vector in turn. In the case of negative-sampling, the two nodes could be used for positive pairs to get the gradient. To measure the *Sensitivity* for embedding models, we outline an initial gradient-based measure, which we call GRAPH-GD. Again, $W \in \mathcal{R}^{|V| \times k}$ are learned latent representations, and $\vec{w}_i \in \mathcal{R}^k$ is the embedding vector of $v_i \in V$ from W.

$$\textbf{GRAPH-GD}(v_b, W, \mathcal{N}, \mathcal{L}) = \text{MEAN}\left\{ \left| \frac{\partial \mathcal{L}(v_b, v_i)}{\partial \vec{w}_i} \right|, v_i \in \mathcal{N}(v_b) \right\}$$

Here $\mathcal{N}$ returns neighbors given a node and $\mathcal{L}$ is the loss function of the corresponding embedding models, which takes a target (input) node and a context node, v_b, v_i, respectively. $\partial \mathcal{L} / \partial \vec{w}_i$ is a *sensitivity* function to return the magnitude of its gradient w.r.t. any $\vec{w}_i \in W$. For example, the loss function of hiearchical softmax-based embedding methods (e.g., DeepWalk and Struc2Vec) has $\mathcal{L}_{hs}(v_b, v_i) = -log\ f(v = v_i | v_b)$, where f is a learned function from the embedding model to have the likelihood of v_i given v_b to appear. In the negative sampling-based embedding methods such as LINE and Node2Vec, their loss functions can be generalized as $\mathcal{L}_{ns}(v_b, v_i) =$

[1]While learning node embeddings such as DeekWalk and LINE, the magnitude of gradient updates do not converge to very small values. This is due to the fact that while relative positions of nodes converge, absolute locations of nodes in the embedding space never converge in practice because of the pair-wise update nature of hierarchical softmax or negative sampling [58, 106].

$-log\ \sigma(\vec{w}_i^T\vec{w}_b) - \sum_{v_z \in V_{neg}} log\ \sigma(-\vec{w}_z^T\vec{w}_b)$, where V_{neg} is a sampled negative node-set. This means that our GRAPH-GD can be applied to most hiearchical softmax and negative sampling-based embedding methods. The magnitudes are aggregated over neighbors of v_b, and, in this work, we uniformly sample a fixed-size (ψ) set of neighbors for v_b, instead of using full neighborhood sets for $\mathcal{N}(v_b)$ above. The theoretical relationship between GRAPH-GD and bridge nodes is described in Section 5.4.1.

While GRAPH-GD enables finding bridge nodes by averaging the magnitudes of gradient updates with a potential bridge node v_b and one of neighboring (context) nodes $v_i \in \mathcal{N}(v_b)$, it does not consider the relative impact of each neighboring node v_i to v_b. In other words, by measuring how the neighboring nodes are differently placed on the embedding space, compared to both the potential bridge node v_b and other nodes in the corresponding cluster, which is being bridged by v_b, we can aggregate them adaptively to find better *bridgeness*-based importance. From the intuition above, we propose our gradient-based measure, GRAPH-wGD:

$$\textbf{GRAPH-wGD}(v_b, W, \mathcal{N}, \mathcal{L}) = \text{MEAN}\left\{ h(v_b, v_i) \cdot \left| \frac{\partial \mathcal{L}(v_b, v_i)}{\partial \vec{w}_i} \right|, v_i \in \mathcal{N}(v_b) \right\},$$

where the weight function $h(v_b, v_i) = 1 + \overline{cos}(\vec{w}_b - \vec{w}_i, -\frac{\partial \mathcal{L}(v_b, v_i)}{\partial \vec{w}_i})$ and $\overline{cos}$ returns the cosine similarity value if the value is positive, otherwise 0. The theoretical analysis in Section 5.4.2 shows how GRAPH-wGD improves GRAPH-GD.

5.3.1 Algorithm

Algorithm 2 shows our procedure to discover m important nodes who have bridging roles. Given the learned embedding model and other inputs, GRAPH-wGD is used for computing node-level importance, and top m nodes are returned.

5.3.2 Complexity Analysis

The time complexity of GRAPH-GD and GRAPH-wGD depends on the number of vertices $|V|$, the computation of gradient, and the size (ψ) neighbor sampling set $\mathcal{N}$. The computation of gradient (as in [106, 107]) in the Skip-Gram architecture corresponds to the size of the embedding vector's dimension k, so the complexity is $O(k\psi|V|)$. Because k and ψ are constants, the complexity can be reduced to $O(|V|)$. Note that we set $\psi = 100$ in experiments.

5.4 Theoretical Analysis

5.4.1 Relationship between GRAPH-GD and Bridge Nodes

Let v_i be the context node of v_b in Skip-Gram architecture, and $\vec{w}_i$ is the embedding of v_i. $\vec{\hat{w}}_i$ be the expected embedding in the latent space to predict v_i given v_b with the perfect probability. We assume that $\left|\frac{\partial \mathcal{L}(v_b,v_i)}{\partial \vec{w}_i}\right| = d(\vec{w}_i, \vec{\hat{w}}_i|\vec{w}_b)$, where the distance function d is to measure the distance between $\vec{\hat{w}}_i$ and $\vec{w}_i$ given the source node's embedding $\vec{w}_b$ for updating $\vec{w}_i$. The assumption is still applicable for both hierarchical softmax and negative samplings-based embeddings. The distance is used for deciding the error to update the embedding vector for $\vec{w}_i$. In order to show how GRAPH-GD finds bridge nodes, we need to assume the learned embedding is "Good Embedding" as follows:

Algorithm 2: Procedure to find Top m important nodes using GRAPH-wGD

```
  Input: A set of vertices $V$, Neighbor function $\mathcal{N}$;
  Input: Embeddings $W$, Loss function $\mathcal{L}$ from a trained embedding model,
         The number of top nodes to return $m$;
1 Init an array, $I$, which has $|V|$ elements. ;
2 Set $I$[j]=GRAPH-wGD($v_j, W, \mathcal{N}, \mathcal{L}$) for $\forall v_j \in V$ ;
3 Return indices who have top $m$ highest scores in $I$;
```

Definition 7 (Good Embedding)**.** *Let W be an embedding learned from a graph G, and a set of clusters C is given. We define W as a "good embedding" when it satisfies the following condition, $d(\vec{w}_i, \vec{\bar{w}}_i, |\vec{w}_b) < d(\vec{w}_k, \vec{\bar{w}}_k, |\vec{w}_b)$ where $\vec{w}_i$, $\vec{w}_b$, and $\vec{w}_k$ are the embeddings of v_i, v_b, and v_k. Here $v_i, v_b \in C_b$ and $v_k \in C_{n(\neq b)}$.*

Based on the definition above, the following shows that GRAPH-GD enables the identification of bridge nodes v_b, compared to any node v_c who has the same degree but less *bridgeness*. Notations (W, $\mathcal{N}$, $\mathcal{L}$) in GRAPH-GD do not change in this analysis, so they are dropped below.

Lemma 3. *Let G be a graph with a set of clusters C and associated embedding W. Let $v_b \in C_b$ be a bridge node and $v_c \in C_b$ be a node with the same degree but fewer inter-community edges (thus, lower bridgeness) than v_b. Suppose that W is a "good embedding" as in Definition 7. Then if GRAPH-GD considers all edges without sampling, GRAPH-GD(v_b) $>$ GRAPH-GD(v_c).*

5.4.2 Comparison between GRAPH-wGD and GRAPH-GD

Def. 8, Def. 9, and Lemma 4 describe how GRAPH-wGD works. Then, Lemma 5, Lemma 6, and Theorem 3 show that GRAPH-wGD can find nodes who have higher *bridgeness* in a larger gap than GRAPH-GD. First, we introduce the definition of *support node.*

Definition 8 (Support Node)**.** *Assume that in graph G with clusters C, v_b is a bridge node. Let B-Cluster(v_b) be the set of clusters that are bridged by v_b. Then let cluster $C_i \subset$ B-Cluster(v_b) and let $\vec{w}_b$ be the embedding of v_b, $\vec{w}_{C_i}$ be the centroid of nodes in C_i in the embedding space, and $\vec{w}_b$ be on a plane π_b. The plane π_b is perpendicular to $\vec{w}_b - \vec{w}_{C_i}$. If both $\vec{w}_{C_i}$ and the embedding ($\vec{w}_i$) of $v_i \in C_i$ are placed over π_b, then v_i is a support node for giving v_b the bridging role for C_i.*

Figure 5.2a shows the illustration of *support nodes.* If both a subset of nodes in C_i and *Centroid*(C_i) are over the plane π_b, then nodes (v_i) in the subset are support

(a) Example with Support Nodes. (b) Gradient update in Lemma 4

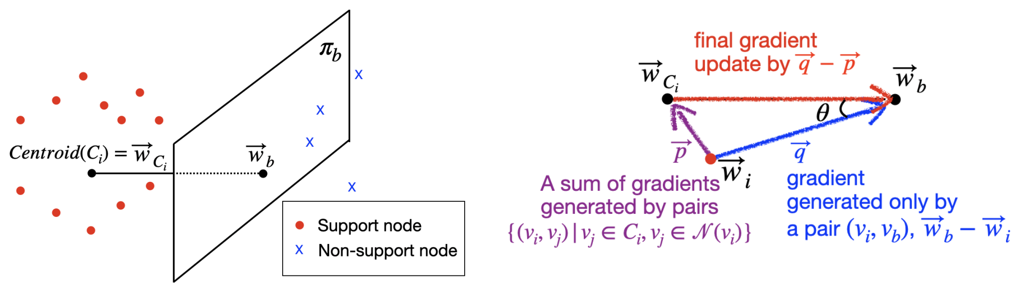

π_b
$Centroid(C_i) = \vec{w}_{C_i}$
$\vec{w}_b$
● Support node
x Non-support node
final gradient update by $\vec{q} - \vec{p}$
$\vec{w}_{C_i}$
$\vec{w}_b$
θ
$\vec{p}$
$\vec{w}_i$
$\vec{q}$
A sum of gradients generated by pairs $\{(v_i, v_j) \mid v_j \in C_i, v_j \in \mathcal{N}(v_i)\}$
gradient generated only by a pair (v_i, v_b), $\vec{w}_b - \vec{w}_i$

Figure 5.2.: Illustration of Lemma 4. Best viewed in Color.

nodes, which are colored red. Note that $\vec{w}_b$ is on π_b and is perpendicular to $\vec{w}_b - \vec{w}_{C_i}$. The support nodes provide more meaningful evidences to identify the bridge role of $\vec{w}_b$ for C_i. than non-support nodes.

We expect that identifying *support nodes* is helpful to measure the *bridgeness*-based importance because they are placed closer to both $\vec{w}_b$ and $\vec{w}_{C_i}$ on the embedding space, so that they can provide more meaningful evidence about the relationship between v_b and C_i, which is potentially helpful to enhance GRAPH-GD. Now we define the weighting method in GRAPH-wGD. In the weighting method, (1) $\vec{w}_i$'s gradient update with v_b and v_i and (2) their difference vector, $\vec{w}_i - \vec{w}_b$, on the embedding space is used for deciding the weight values. The formal definition of directional weight is given as:

Definition 9 (Directional Weight)**.** *Let v_b be a bridge node and v_i be one of context nodes from v_b. $\vec{w}_b$ is the embedding of v_b. $\vec{w}_b$ and $\vec{w}_i$ are the embeddings of v_b and v_i, respectively. $-\frac{\partial \mathcal{L}(v_b, v_i)}{\partial \vec{w}_i}$ represents the gradient update of $\vec{w}_i$ using v_b and v_i. We define a directional weight function, $h(v_b, v_i) = 1 + \overline{cos}\left(\vec{w}_b - \vec{w}_i, -\frac{\partial \mathcal{L}(v_b, v_i)}{\partial \vec{w}_i}\right)$, where $\overline{cos}$ returns the cosine similarity value if the value is positive, otherwise 0.*

Based on Definition 9 above, the following Lemma shows that the directional weight is able to find support nodes.

Lemma 4. *Let v_b be a bridge node and B-Cluster(v_b) is a set of clusters that are bridged by v_b. Assume there is a cluster $C_i \subset$ B-Cluster(v_b) where the nodes in C_i are densely connected to each other. If v_i is a support node for giving v_b the bridging role for C_i, then $h(v_b, v_i) > 1$. If v_i is a not support node for C_i and v_b, then $h(v_b, v_i) = 1$.*

Using Definition 8, Definition 9, and Lemma 4 above, we can explain how the weight of GRAPH-wGD works. Again, notations (W, $\mathcal{N}$, $\mathcal{L}$) in GRAPH-wGD are dropped for clarity.

Lemma 5. *Let v_b be a bridge node and B-Cluster(v_b) is a set of clusters that are bridged by v_b. Assume there is a cluster $C_i \subset$ B-Cluster(v_b) where the nodes in C_i are densely connected each other. If v_b is a bridge node and its context nodes include at least one support node, then GRAPH-wGD(v_b) $>$ GRAPH-GD(v_b).*

We can also see how GRAPH-GD compares to GRAPH-wGD with nodes that have no support node. Nodes with no support node mean that their context nodes are placed in the other direction against the centroid of the cluster connected. Thus, the node has no bridging role to the corresponding cluster.

Lemma 6. *When context nodes of a node $v_c \in V$ do not include support nodes, GRAPH-GD(v_c) $=$ GRAPH-wGD(v_c).*
The following theorem shows that the directional weight of GRAPH-wGD helps to find nodes who have higher *bridgeness* in a larger gap than GRAPH-GD.

Theorem 3. *Let G be a graph with a set of clusters C and associated embedding W. We denote $v_b \in C_b$ be a bridge node and $v_c \in C_b$ be a node with the same degree but fewer inter-community edges (thus, lower bridgeness) than v_b. Assume that the context nodes of v_b include at least one support node, and the context nodes of v_c do not include a support node. If W is a "good embeddings" (Def. 7), nodes in C_b are densely connected with each other, and GRAPH-wGD and GRAPH-GD consider all edges without sampling, then GRAPH-wGD(v_b) $>$ GRAPH-GD(v_b) $>$ GRAPH-GD(v_c) $=$ GRAPH-wGD(v_c).*

From the Theorem above, now we can expect GRAPH-wGD to assign importance scores that correlate more closely with *bridgeness*.

5.5 Experimental Settings

5.5.1 Data: Real-world Network Data

To evaluate our method, five real-world networks are used. First, Wikipedia (Wiki) [108] is a co-occurrence network of words appearing in a Wikipedia dump, which is composed of 4,777 nodes, 184,812 edges, and 40 different labels. Second, BlogCatalog [109] is a network of social relationships of the bloggers listed on the BlogCatalog website. The network has 10,312 nodes, 333,983 edges, and 39 different labels. Third, Enron network data [110] is composed of 150 nodes and 1,853 edges (after unifying 517,431 emails). For class labels, employees in management positions are used and they are also used for evaluating our method. Fourth, NeurIPS dataset is pre-processed from a Kaggle repository (benhamner/nips-papers), and co-authorship edges are extracted from their papers between 1987 and 2017. It has 9,784 vertices and 22,198 edges. NeurIPS is used for visualization and qualitative analysis. Lastly, we also use the Karate Network data for visualization and rank correlation analysis.

5.5.2 Evaluation Measures

Spearman's Rank Correlation The Spearman correlation coefficient is defined as the Pearson correlation coefficient between the rank variables. This is to evaluate how similar the ranking of nodes by the importance scores of other methods is to the ranking by *bridgeness*.

Prediction Change (PC) This measure is to see how much predicted probability is changed in node classification tasks after perturbing edges of important nodes. For this, top $k\%$ of important nodes from each method are perturbed as in Def. 5. The final score is computed by the sum of the absolute difference in probabilities of all class labels after perturbing the graph and relearning the embedding. For this, Wiki, BlogCatalog, and Enron, whose labels are available, are used.

Emergence (EM) When we perturb more important node(s), we expect the nearest neighboring nodes of each node to be changed more. To measure how much the new neighbors are newly appeared after perturbing edges of v_j, we use the following measure, *Emergence* as:

$$(New\ neighbors')\ Emergence(v_j) = 1 - \frac{|N_{v_j} \cap N_{v_j}^{perturb}|}{|N_{v_j}|},$$

where N_{v_j} is the number of n-nearest neighbors of a node v_j on the latent space. $N_{v_j}^{perturb}$ is the number of n-nearest neighbors of a node v_j after perturbation. Therefore, we can measure how much of the neighboring nodes are changed based on the number of nodes shifted on the space. n is set to 5% of $|V|$. The perturb function from Def. 5 is used.

5.5.3 Comparison Models to Determine Node Importance

Bridgeness We first learn node embedding and then capture nodes' *bridgeness* scores, which is based on found clusters on the learned embedding using spectral clustering. Here *bridgeness* is defined as in Equation (5.1).

Degree Degree is leveraged to find important nodes.

Personalized PageRank (PPR) PPR is also used as a baseline method. We use default parameters in NetworkX 2.1.

Greedy The sum of absolute changes in predictions over all labels from an input graph G and a perturbed graph G'_{-v_i} by v_i is leveraged for the importance score of v_i as in the Equation (2.6). The perturbation strategy still follows the same Def. 5.

LIME LIME [14] measures how much nearby nodes are locally important to predict the label of a node. We use the same learning parameters as in [14]. The final importance for a node v_i is from the averaged importance of the neighboring nodes, $\{v_c \in \mathcal{N}(v_i)\}$.

GNNExplainer We use the same objective function of GNNExplainer [17] to quantify the importance of each node. Thus, the mutual information between predictions from an input graph and a perturbed graph by v_i is leveraged for the importance score of v_i. In this case, we assume that nodes who return low mutual information value are more important.

5.5.4 Experimental Setting

For learning DeepWalk for Wiki and BlogCatalog, we set all the parameters as in [7] with hierarchical softmax because they experimented with the same datasets. For Enron and Karate networks, we set its random walk size as 5, the window size = 5, and the size of embedding vector = 8. For NeurIPS, all parameters are set as in the Wiki dataset. For learning LINE, we also use the same sizes of embedding vector. Learning rates for both embeddings are set as the original papers for all datasets. To get the magnitude of gradient on LINE, the gradient of 1st-order proximity embedding is exploited because they don't have joint loss function. For the perturbation ratio, we set $\alpha = 0.4$. When we perturb edges, clusters are found from spectral clustering on the learned embedding and the number of clusters is given from the number of their class labels. For classification, a fully-connected neural network with 2 layers is used for multi-label or single-label prediction. The hidden node size is defined as two times of the class label size of each dataset. ADAM is chosen for optimization and learning weight is 1e-5. Regarding node-set splits, top 40% of high-degree nodes are chosen for a testing node-set (i.e., for rank correlation, PC, and EM). 10% of nodes from the remaining node-set are randomly selected for training, and another 10% are leveraged for validation. We note that the training and validation node-sets

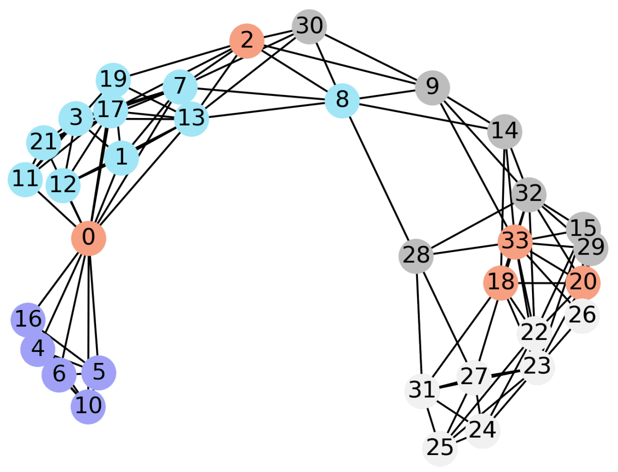


Figure 5.3.: Top 5 Important Nodes in DeepWalk embedding by GRAPH-wGD. Visualization is from t-SNE. Each color represents cluster ID after spectral clustering except oranges.

are used only for Greedy, LIME, and GNNExplainer, and the results are averaged after 3 different random training/validation node-set selections.

5.6 Experimental Results

5.6.1 Experiment 1: Visualization and Rank Correlation Analysis with Karate Graph

In this experiment, Zachary's Karate club network is used to show how the ranking from our methods, GRAPH-GD and GRAPH-wGD, work by measuring rank correlation to the ranking from *Bridgeness* and visualizing top nodes for qualitative analysis. Table 5.1 shows the result of Spearman's rank correlation coefficient (ρ)

Table 5.1.: Spearman's Rank Correlation in Karate, Wiki, and BlogCatalog. (Bold means the best score.)

	Karate		Wiki		BlogCatalog	
	DW	LINE	DW	LINE	DW	LINE
Greedy	0.175	0.350	0.160	0.056	0.140	0.010
LIME	0.175	0.359	0.177	0.072	0.185	0.141
GNNExplainer	0.175	0.350	0.193	0.098	0.196	0.160
GRAPH-GD	0.634	0.607	0.197	0.158	0.268	0.141
GRAPH-wGD	**0.683**	**0.634**	**0.392**	**0.458**	**0.368**	**0.334**

Table 5.2.: Prediction Change (PC) after the Perturbation of Top $3, 5, 7\%$ Nodes in Wiki. (Bold means the best score and *Bridgeness* provides theoretical upper bounds.)

	DeepWalk			LINE		
	3%	5%	7%	3%	5%	7%
Bridgeness	2.03	1.65	1.84	37.23	38.38	38.88
Degree	1.00	1.46	1.41	25.28	25.55	27.44
PPR	1.12	1.04	1.17	26.45	26.86	28.00
Greedy	1.11	1.06	1.14	29.30	29.99	33.61
LIME	1.15	1.10	1.19	25.34	32.41	37.84
GNNExplainer	1.35	1.37	1.42	25.94	26.57	30.63
GRAPH-GD	1.45	1.09	1.25	24.31	26.94	34.06
GRAPH-wGD	**1.98**	**1.59**	**1.68**	**34.25**	**35.88**	**43.14**

Table 5.3.: Prediction Change (PC) after the Perturbation of Top $3, 5, 7\%$ Nodes in BlogCatalog. (Bold means the best score and *Bridgeness* provides theoretical upper bounds.)

	DeepWalk			LINE		
	3%	5%	7%	3%	5%	7%
Bridgeness	2.73	2.76	2.75	41.29	40.92	38.95
Degree	2.32	2.15	2.29	37.54	36.12	38.09
PPR	2.30	2.03	2.27	38.72	39.10	37.62
Greedy	2.15	2.27	2.19	34.87	38.19	36.52
LIME	2.20	1.98	2.33	37.07	38.14	38.65
GNNExplainer	2.48	2.54	2.71	37.50	37.73	38.38
GRAPH-GD	**2.60**	2.34	**2.77**	37.42	37.78	37.17
GRAPH-wGD	2.57	**2.74**	2.76	**40.73**	**40.98**	**39.48**

using two node embeddings, DeepWalk (DW) and LINE. In both embeddings, our GRAPH-wGD shows the best performance in finding bridge nodes, and p-values for our methods were less than 0.05.

In Figure 5.3, we can see the visualization of the Karate network with the top 5 important nodes that are found by GRAPH-wGD. Node 0, 2, 33, 18, and 20 are chosen as important nodes by GRAPH-wGD, and they are serving as bridge nodes on the learned latent space. The result also corresponds to bridge nodes found in other studies (e.g., [97]).

Table 5.4.: Emergence (EM) after the Perturbation of
Top 3, 5, 7% Nodes in Wiki. (Bold means the best score and *Bridgeness* provides theoretical upper bounds.)

	DeepWalk			LINE		
	3%	5%	7%	3%	5%	7%
Bridgeness	0.706	0.705	0.707	0.644	0.659	0.665
Degree	0.260	0.295	0.290	0.375	0.399	0.411
PPR	0.284	0.284	0.216	0.381	0.405	0.421
Greedy	0.544	0.572	0.513	0.549	0.574	0.591
LIME	0.514	0.527	0.584	0.539	0.582	0.611
GNNExplainer	0.680	0.679	0.674	0.548	0.563	0.572
GRAPH-GD	0.682	**0.689**	0.691	0.523	0.534	0.555
GRAPH-wGD	**0.689**	**0.689**	**0.697**	**0.564**	**0.590**	**0.630**

Table 5.5.: Emergence (EM) after the Perturbation of Top 3, 5, 7% Nodes in BlogCatalog. (Bold means the best score and *Bridgeness* provides theoretical upper bounds.)

	DeepWalk			LINE		
	3%	5%	7%	3%	5%	7%
Bridgeness	0.682	0.642	0.668	0.624	0.628	0.632
Degree	0.422	0.475	0.504	0.346	0.351	0.359
PPR	0.328	0.350	0.383	0.343	0.349	0.357
Greedy	0.508	0.509	0.527	0.498	0.510	0.518
LIME	0.463	0.510	0.521	0.484	0.510	0.523
GNNExplainer	0.443	0.521	0.546	0.497	0.509	0.517
GRAPH-GD	0.639	**0.654**	0.659	0.487	0.488	0.502
GRAPH-wGD	**0.650**	0.651	**0.661**	**0.510**	**0.518**	**0.541**

5.6.2 Experiment 2: Correlation Analysis and Measurement of the Impact of Found Important Nodes in Wiki and BlogCatalog

In this experiment, we evaluate the ranking from our methods by getting the rank correlation to *Bridgeness*-based ranking and verify the impact of found important nodes w.r.t. changes in label prediction and neighboring nodes.

Table 5.1 shows Spearman's Rank Correlation to Bridge Nodes. In the experiments, our GRAPH-wGD outperforms other methods. In particular, GRAPH-GD works better than other alternatives in rank correlation, but our GRAPH-wGD showed better performance than the GRAPH-GD. This indicates that capturing support nodes helps to find bridge nodes. Other prediction probability-based methods such as Greedy and GNNExplainer are not much correlated to *Bridgeness*-based ranking. GNNExplainer shows better correlations than other baselines, but the correlations are still smaller than GRAPH-wGD's result.

Table 5.2 and 5.3 show the changes in prediction after the graph is perturbed and the embedding is relearned. In the tables, 3, 5, and 7% of important nodes, which are found from each method, are perturbed to see the effect w.r.t. the change of prediction. Comparing to LINE, DeepWalk shows relatively smaller changes because the two datasets are relatively dense and the characteristics of random walks diminish the effect of perturbation. Here we note that *Bridgeness* provides empirical upper bounds to all methods as we expected from Theorem 2. However, we note that it is computationally intensive to calculate *Bridgeness*, and the top nodes identified by GRAPH-wGD more efficiently perform similarly.

In Table 5.4 and 5.5, *emergence* (EM) is reported to show how many new neighbors appear in a node's neighborhood after perturbing each method's 3, 5, and 7 % top nodes and relearning the embedding. Again, GRAPH-wGD shows more changes in *Emergence* than other alternatives.

5.6.3 Experiment 3: Finding Important Nodes and Measuring the Impact of Found Nodes in Enron

In this experiment, we explore how the discovered important nodes are associated with organizational positions. By leveraging the known position information of each person in the Enron dataset, we first measure how much of the importance score from each method is effective to find people who are in management positions. For this, we regard management people as important and find the precision@k% from the importance ranking for each method. We note that GRAPH-wGD shows high rank correlations (>0.6). In Table 5.6, the precision@k(=20% nodes) results are reported.

The result shows that our GRAPH-wGD is also effective to find management people. This indicates that the management roles are also highly correlated to bridging roles in communication.

Table 5.7 shows the top important people from *Degree* and GRAPH-wGD. While *Degree* finds globally important people such as the CEO and Chairman, GRAPH-wGD finds more locally important nodes like people who worked as VP and managers. This also implies that identifying bridging people helps to understand the data and the learned embedding.

For Enron dataset, we also investigate the impact of found nodes. As in the Table 5.8, GRAPH-wGD is able to find more prediction changes and *emergence*s when their top nodes are perturbed. In particular, the *emergence* in Deepwalk embedding shows significantly better results than other alternatives.

5.6.4 Experiment 4: Visualization with NeurIPS

On the NeurIPS data, we report the top eight discovered important people in Table 5.9. For GRAPH-wGD with DeepWalk, it finds many interdisciplinary researchers such as R. Rosales (Machine Learning Scientist who works on Text Mining, Computer Vision, Graphics, and Medicine) and I. Guyon (Machine learning consultant who works on machine learning theory, handwriting recognition, and biomedical research for genomics, proteomics, cancer). The result is also visualized in Figure 5.4. The visualization is from t-SNE algorithm with the default parameters in scikit-learn.

Table 5.6.: Precision@20% in finding management nodes in Enron. (Bold means the best score.)

	DeepWalk	LINE
Greedy	0.25	0.30
LIME	0.40	0.45
GNNExplainer	0.45	0.30
GRAPH-GD	0.45	0.45
GRAPH-wGD	**0.60**	**0.75**

Table 5.7.: Top 5 nodes in Enron

	Top 5 Nodes
Degree	L. Taylor, S. Beck, J. Lavorato (CEO), K. Lay (Chairman), L. Kitchen (President)
GRAPH-wGD	S. Scott, F. Sturm (VP), J. Williams (VP), K. Lay (Chairman), P. Allen (Manager)

Table 5.8.: Prediction Change (PC) and Emergence (EM) in Enron. (Bold means the best score and *Bridgeness* provides theoretical upper bounds.)

	PC				EM			
	DeepWalk		LINE		DeepWalk		LINE	
	30%	40%	30%	40%	30%	40%	30%	40%
Bridgeness	7.14	7.34	0.13	0.12	0.574	0.578	0.085	0.081
Degree	6.35	6.49	0.10	0.10	0.359	0.361	0.002	0.002
PPR	5.75	5.73	0.10	0.10	0.369	0.370	0.005	0.004
Greedy	3.13	3.96	0.08	0.08	0.420	0.424	0.052	0.050
LIME	0.51	0.45	0.08	0.08	0.061	0.061	0.053	0.053
GNNExplainer	3.39	3.43	0.10	**0.10**	0.440	0.439	0.053	0.056
GRAPH-GD	4.71	4.68	0.10	0.09	**0.536**	**0.536**	0.083	0.079
GRAPH-wGD	**6.75**	**6.72**	**0.11**	**0.10**	0.526	0.526	**0.085**	**0.086**

Table 5.9.: Top 8 nodes in NeurIPS

	Top 8 Nodes
Degree	G. Hinton, Y. Bengio, M. Jordan, Z. Ghahramani, K. Muller, B. Scholkop, A. Ng, R. Salakhutdinov
GRAPH-wGD	R. Rosales , I. Guyon , M. Figueired, C. Scott, M. Zibulevsky, A. McCallum, E. Xing, R. Vogelstein

Colors in the Figure represent Cluster IDs by K-Means on the learned embeddings. Names of the top 8 people are placed in their positions, and they are located near the center or boundaries of clusters. This indicates that the people are likely to have bridging roles across research communities. Researchers such as A. McCallum and E. Xing are also identified in the central positions because they have contributed to the foundation of ML/NLP while collaborating in many application domains. Surprisingly, the top *Degree* nodes do not overlap with the top nodes of GRAPH-wGD.

This indicates that while they may be the most prolific researchers, they have less global impact on the learned embedding, particularly if they publish only with a small number of collaborators in the same area. Figure 5.5 also shows the Top 8 people who

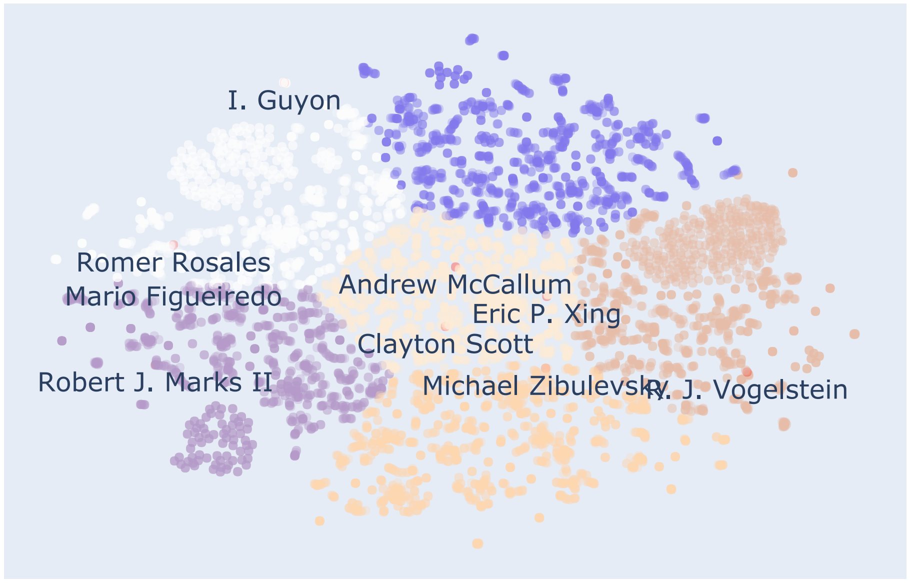


Figure 5.4.: Visualization of Top 8 important authors, which are found by GRAPH-wGD for the NeurIPS dataset.

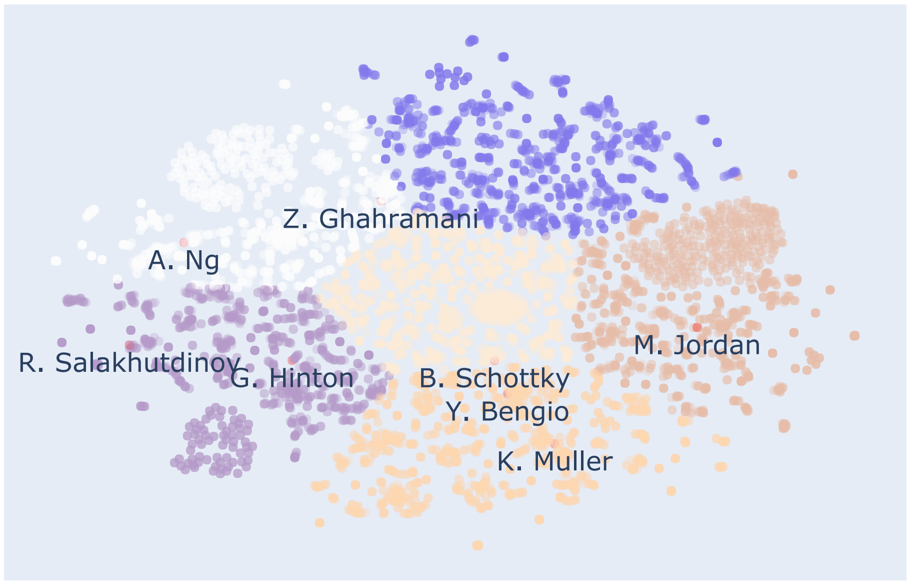


Figure 5.5.: Visualization of Top 8 important authors, which are found by Degree for the NeurIPS dataset.

Table 5.10.: Ablation study: Spearman's rank correlation

	Wiki		BlogCatalog	
	DeepWalk	LINE	DeepWalk	LINE
GRAPH-GD	0.197	0.158	0.268	0.141
GRAPH-wGD (+ and -)	0.292	0.122	0.195	0.181
GRAPH-wGD (abs)	0.189	0.197	0.122	0.225
GRAPH-wGD	**0.392**	**0.458**	**0.368**	**0.334**

Table 5.11.: Prediction Change (PC) and Emergence (EM) after the perturbation of Top 3% nodes in Wiki and BlogCatalog.

	Wiki				BlogCatalog			
	PC		EM		PC		EM	
	DW	LINE	DW	LINE	DW	LINE	DW	LINE
GRAPH-GD	1.45	24.31	0.682	0.523	**2.60**	37.42	0.523	0.487
GRAPH-wGD (+ and -)	1.36	30.83	0.685	0.561	2.35	37.98	0.561	0.508
GRAPH-wGD (abs)	1.08	32.79	0.665	0.525	1.97	35.69	0.525	**0.511**
GRAPH-wGD	**1.98**	**34.25**	**0.689**	**0.564**	2.57	**40.73**	**0.564**	0.510

have the highest degrees by using the same t-SNE visualization. Different from the results of GRAPH-wGD, many of them are located toward the center of local clusters rather than the boundary. For example, G. Hinton, M.Jordan, and Y. Bengio are located in the central positions at purple, red, and orange clusters, respectively. This implies that they take more community-core-like roles in the learned embedding.

5.7 Ablation Study and Runtime Analysis

5.7.1 Ablation Study

In our proposing GRAPH-wGD, we might have used other weighting functions by changing our assumptions w.r.t support nodes. In this section, we compare different variants of GRAPH-wGD on Wiki and BlogCatalog datasets. Here, the variants include modification to the cosine similarity function in GRAPH-wGD:

$$\textit{GRAPH-wGD (+ and -):}\ h(v_b, v_i) = 1 + cos\left(\vec{w}_b - \vec{w}_i, -\frac{\partial \mathcal{L}(v_b, v_i)}{\partial \vec{w}_i}\right),$$
$$\textit{GRAPH-wGD (abs):}\ h(v_b, v_i) = 1 + \left|cos\left(\vec{w}_b - \vec{w}_i, -\frac{\partial \mathcal{L}(v_b, v_i)}{\partial \vec{w}_i}\right)\right|.$$

GRAPH-wGD (+ and -) let negative angular distance offset the importance when aggregating scores, and *GRAPH-wGD (abs)* denotes that negative and positive support nodes are handled in the same way. They are compared to our proposed weighting function in Section 5.3.1. We note that our function leverages positive support nodes only. Table 5.10 and 5.11 show results on Spearman's Rank Correlation, Prediction Change (PC), and Emergence (EM). In the results, GRAPH-wGD (+ only), our proposed method, shows better results in most settings. This implies that support nodes help to identify bridging nodes and have more impact on changes in prediction and local neighbors. Moreover, Table 5.10 shows that our choice of GRAPH-wGD is better than GRAPH-GD and all other variants in Spearman's Rank Correlation. The observation also indicates that identifying support nodes improves the identification of bridge nodes.

5.7.2 Runtime Comparison

In Table 5.12, we report the runtime of our method on Wiki and BlogCatalog datasets. Our method was implemented in Python and was executed on an Intel

Table 5.12.: Elapsed time in Wiki and BlogCatalog

	Wiki		BlogCatalog	
	DeepWalk	LINE	DeepWalk	LINE
Bridgeness	748.41 sec	748.41 sec	10,039.59 sec	10,039.59 sec
Greedy	214.32 hours	28.72 hours	902.24 hours	68.57 hours
LIME	156.69 days	12.65 days	582.56 days	381.03 days
GNNExplainer	192.20 hours	29.78 hours	850.03 hours	70.49 hours
GRAPH-wGD	**23.30 sec**	**67.41 sec**	**141.93 sec**	**233.44 sec**

Xeon Gold 6126 CPU@2.60GHz server with 192GB RAM. The reported running time is measured by assuming that the code is run by a single process. We note that all the methods are executed by multiple processes and merged in the actual computations. As in Table 5.12, our algorithm scales greatly, compared to other alternatives. In particular, *Greedy*, LIME, and GNNExplainer need to re-learn node embedding every time they need perturbation. Thus, in the case of DeepWalk with the Skip-Gram Architecture, which takes $O(|V|log|V|)$, and *Greedy* and GNNExplainer take $O(|V|^2log|V|)$ due to the node-wise perturbations. Similarly, LIME takes $O(|V|^3log|V|)$ to consider additional local neighbor perturbations. For *Bridgeness*, we run both embedding and eigenvector decomposition and it takes $o(|V|^{2.367})$. Therefore, it takes a large amount of time. Meanwhile, our method does not need to learn the graph again and takes only $O(|V|)$.

5.8 Concluding Remarks

We have demonstrated the theoretical relationship between node embeddings including DeepWalk and LINE and *bridgeness* and have proposed a new algorithm, GRAPH-wGD, for efficiently measuring *bridgeness*-based importance to generate explanations to learned embeddings. In particular, our GRAPH-wGD shows superior performance than other alternatives on 5 different datasets.

6 CONCLUSION

While previous work in semi-supervised learning provides many promising results, it is still limited in leveraging complex interaction patterns and local inferences over high-order paths. In this dissertation, we first proposed a graph neural network architecture to leverage local inferences with labels over high-order paths. While label propagation has been widely used to improve the performance for semi-supervised learning, existing deep neural networks have not much explored the property. In this work, we described REGNN, a graph neural network architecture that exploits label propagation in a message-passing framework. REGNN not only learns from high-order path information but also adaptively aggregates the hidden representation of neighboring nodes using role equivalence. In our experimental results, REGNN consistently showed improved performance for semi-supervised node classification in graphs without node attributes. Specifically, REGNN produces significant gains compared to the best competitor (GCN)—reducing classification error up to 26% and an average of 5.4% for all label proportions/datasets.

The second contribution is a neural network architecture (TSGNet) which can learn jointly from static and temporal neighborhood structure. The architecture exploits the interactions among local neighbors over time, by learning the temporal evolution of a low-dimensional embedding from a GCN, and models its static neighborhood with a densely connected NN. TSGNet is able to improve classification performance by utilizing both patterns in social interactions over time and the set of nodes in the aggregate relational neighborhood. Experimental results show that TSGNet reduces classification error up to 24% across three real-world social interaction datasets that are composed on temporal graph snapshots, compared to state-of-the-art models. Moreover, compared to the best competitor (GraphSAGE), TSGNet with or w/o attributes produces an average reduction in classification error of 10%.

Finally, in the third component of this dissertation, the theoretical relationship between node embeddings including DeepWalk and LINE and *bridgeness* is explored, and we have proposed a new algorithm, GRAPH-wGD, for efficiently measuring *bridgeness*-based importance to generate explanations for learned embeddings. This allows us to have a better understanding of representation learning by providing globally important nodes.

6.1 Theoretical and Empirical Contribution

We summarize the theoretical and empirical contributions of this dissertation below:

- Models and Frameworks
  - Development of REGNN — a semi-supervised node classification method for supporting collective inference and leveraging higher-order paths to consider more distant neighbors in static graphs
  - Development of TSGNET — a semi-supervised node classification method to learn over temporal interaction graphs
  - Development of perturbation-based importance scoring framework to generate explanations of learned node embeddings
- Algorithmic Contributions
  - Development of importance sampling-based mini-batch learning methods for REGNN and TSGNET
  - Development of role equivalence-based attention mechanism for REGNN
  - Development of GRAPH-wGD measure to efficiently find global importance of nodes on the learned embedding.
- Theoretical Contributions

  - Theoretical study about the relationship between GCN and Label propagation when the same observed label input is given
  - Investigation about the theoretical connection between node importance with respect to the impact on node embedding and bridge scores of nodes in graphs
  - Analysis on the effect of the weighting function and the gradient magnitude in GRAPH-wGD measure with respect to approximating bridgeness scores of nodes

- Evaluation
  - Development of several evaluation criteria to explore the utility of the explanations found for learned node embeddings

6.2 Future Work

There are several directions to pursue after the completion of this dissertation. First, we can improve the label propagation in GNNs using different types of walks. For example, labels can be propagated by leveraging newly generated edges that are found from structural similarities or role similarity in latent space. This information provides complementary information with existing direct edges, which can be used to model more complex interactions.

Second, we can explore more theoretical properties on TSGNet. TSGNet currently learns both static and temporal aspects at the same time, and it is still not clear which temporal aspects are actually learned or which patterns are being considered more favorably. In addition, other pre-training strategies (e.g., [111]) can be studied to overcome temporal sparsity across temporal graphs. Because GNNs often suffer from less amount of labeled data and fewer domain-specific features, the lack of temporal interactions may bother to learn TSGNet. Moreover, the effect of randomization should be studied more. TSGNet shows similar performance to the same model with randomized graph inputs w.r.t. time window IDs. Therefore, we can explore the effect of the randomization more and attempt to learn temporal graphs by regarding them as a set of sub-graphs using permutation-invariant functions [91].

Third, the findings of GRAPH-wGD give more ideas to enhance the explanations of GRAPH-wGD and generate better explanations for node embeddings. One of the future works is to generate explanations as a sub-graph to learned embeddings. While finding single nodes provides valuable information in small graphs, sub-graphs may help us to give more high-level information in large-scale networks. Moreover, local explanations can be explored instead of global explanations. It will be potentially useful to provide more ego-centric interpretation to a single node-based prediction and decide whether the corresponding node embedding is the right choice or not. Another direction is to leverage observed class labels to have a confidence interval of each node embedding. Because there is no general statistical framework to interpret representation learning, we can propose new statistical testing methods to provide a confidence interval of each node embedding. Supporting other embeddings such as GNNs and Hyperbolic embedding [112] is also one of promising directions. While GNNexplaner [17] reported a preliminary result on GNNs, but it does not provide theoretical explanations and has not evaluated using real-world datasets. From the perspective of evaluation, human participation should also be encouraged to do a more accurate analysis of the generated explanations. Depending on downstream tasks, the global explanations can be evaluated from human participation to confirm whether the explanations are relevant to the tasks. Lastly, the theoretical relationship between node embedding and bridgeness can be used for developing privacy-preserving techniques. In particular, edge-based differential privacy computing can leverage this finding to choose which nodes should be stored on edge and local servers.

REFERENCES

[1] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2014.

[2] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the Web Conference*, 2015.

[3] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. struc2vec: Learning node representations from structural identity. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 385–394, 2017.

[4] Jian Tang, Meng Qu, and Qiaozhu Mei. Pte: Predictive text embedding through large-scale heterogeneous text networks. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1165–1174, 2015.

[5] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the International Conference on Learning Representations*, 2016.

[6] Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the International Conference on Machine Learning*, 2016.

[7] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

[8] 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.

[9] Joseph J. Pfeiffer, III, Jennifer Neville, and Paul N. Bennett. Overcoming relational learning biases to accurately predict preferences in large scale networks. In *Proceedings of the Web Conference*, 2015.

[10] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the Conference on Neural Information Processing Systems*, 2017.

[11] John Moore and Jennifer Neville. Deep collective inference. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 2364–2372, 2017.

[12] Walter Nelson, Marinka Zitnik, Bo Wang, Jure Leskovec, Anna Goldenberg, and Roded Sharan. To embed or not: network embedding as a paradigm in computational biology. *Frontiers in Genetics*, 10, 2019.

[13] Marinka Zitnik, Francis Nguyen, Bo Wang, Jure Leskovec, Anna Goldenberg, and Michael M Hoffman. Machine learning for integrating data in biology and medicine: Principles, practice, and opportunities. *Information Fusion*, 50, 2019.

[14] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

[15] Scott Lundberg and Su-in Lee. A Unified Approach to Interpreting Model Predictions. In *Proceedings of the Conference on Neural Information Processing Systems*, 2017.

[16] Lingyang Chu, Xia Hu, Juhua Hu, Lanjun Wang, and Jian Pei. Exact and Consistent Interpretation for Piecewise Linear Neural Networks: A Closed Form Solution. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2018.

[17] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. Gnnexplainer: Generating explanations for graph neural networks. In *Proceedings of the Conference on Neural Information Processing Systems*, pages 9240–9251, 2019.

[18] Xiaojin Zhu and Zoubin Ghahramani. Learning from labeled and unlabeled data with label propagation. *CMU tech report CMU-CS-02-107*, 2002.

[19] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research*, 7(Nov):2399–2434, 2006.

[20] Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*, pages 639–655. Springer, 2012.

[21] Brian Gallagher, Hanghang Tong, Tina Eliassi-Rad, and Christos Faloutsos. Using ghost edges for classification in sparsely labeled networks. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 256–264, 2008.

[22] Rongjing Xiang and Jennifer Neville. Pseudolikelihood em for within-network relational learning. In *Proceedings of the IEEE International Conference on Data Mining*, pages 1103–1108, 2008.

[23] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the ACM International Conference on Information and Knowledge Management*, 2015.

[24] Cheng Yang, Maosong Sun, Zhiyuan Liu, and Cunchao Tu. Fast network embedding enhancement via high order proximity approximation. In *Proceedings of the International Joint Conferences on Artificial Intelligence*, 2017.

[25] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. Verse: Versatile graph embeddings from similarity measures. In *Proceedings of the Web Conference*, pages 539–548, 2018.

[26] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of the Conference on Neural Information Processing Systems*, 2016.

[27] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*, 2017.

[28] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *Proceedings of the International Conference on Learning Representations*, 2019.

[29] Zhenpeng Zhou and Xiaocheng Li. Graph convolution: A high-order and adaptive approach. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017.

[30] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[31] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *Proceedings of the International Conference on Learning Representations*, 2018.

[32] Kiran K Thekumparampil, Chong Wang, Sewoong Oh, and Li-Jia Li. Attention-based graph neural network for semi-supervised learning. *arXiv preprint arXiv:1803.03735*, 2018.

[33] Yedid Hoshen. Vain: Attentional multi-agent predictive modeling. In *Proceedings of the Conference on Neural Information Processing Systems*, pages 2701–2711, 2017.

[34] Ruth C Fong and Andrea Vedaldi. Interpretable Explanations of Black Boxes by Meaningful Perturbation. In *Proceedings of the International Conference on Computer Vision*, pages 3429–3437, 2017.

[35] Jiwei Li, Will Monroe, and Dan Jurafsky. Understanding neural networks through representation erasure. *arXiv preprint arXiv:1612.08220*, 2016.

[36] Alexander Binder, Sebastian Lapuschkin, Klaus-Robert Muller, Gregoire Montavon, and Wojciech Samek. Evaluating the Visualization of What a Deep Neural Network Has Learned. *IEEE Transactions on Neural Networks and Learning Systems*, 28(11):2660–2673, 2016.

[37] Ninghao Liu, Xiao Huang, Jundong Li, and Xia Hu. On Interpretation of Network Embedding via Taxonomy Induction. In *Proceedings of the ACM SIGKDD International Conference on Know ledge Discovery and Data Mining*, 2018.

[38] Pang Wei Koh and Percy Liang. Understanding Black-box Predictions via Influence Functions. In *Proceedings of the International Conference on Machine Learning*, 2017.

[39] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *Proceedings of the International Conference on Machine Learning*, pages 1321–1330, 2017.

[40] Michael Cogswell, Abhishek Das, and Dhruv Batra. Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization. *Proceedings of the International Conference on Computer Vision*, 2017.

[41] David; et al. Bau. Network Dissection : Quantifying Interpretability of Deep Visual Representations. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2017.

[42] C. C. Jay Kuo, Min Zhang, Siyang Li, Jiali Duan, and Yueru Chen. Interpretable Convolutional Neural Networks via Feedforward Design. *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 8827–8836, 2018.

[43] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning Important Features Through Propagating Activation Differences. In *Proceedings of the International Conference on Machine Learning*, 2017.

[44] Andrew Slavin Ross, Michael C. Hughes, and Finale Doshi-Velez. Right for the right reasons: Training differentiable models by constraining their explanations. In *Proceedings of the International Joint Conferences on Artificial Intelligence*, pages 2662–2670, 2017.

[45] Jianbo Chen, Le Song, Martin J. Wainwright, and Michael I. Jordan. Learning to Explain: An Information-Theoretic Perspective on Model Interpretation. In *Proceedings of the International Conference on Machine Learning*, 2018.

[46] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.

[47] Chunshui Cao, Xianming Liu, Yi Yang, Yinan Yu, Jiang Wang, Zilei Wang, Yongzhen Huang, Liang Wang, Chang Huang, Wei Xu, et al. Look and think twice: Capturing top-down visual attention with feedback convolutional neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2956–2964, 2015.

[48] Jiwei Li, Xinlei Chen, Eduard Hovy, and Dan Jurafsky. Visualizing and understanding neural models in nlp. *arXiv preprint arXiv:1506.01066*, 2015.

[49] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Proceedings of the Conference on Neural Information Processing Systems*, pages 2654–2662, 2014.

[50] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[51] Jie Zhu, Ying Shan, JC Mao, Dong Yu, Holakou Rahmanian, and Yi Zhang. Deep embedding forest: Forest-based serving with deep embedding features. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1703–1711, 2017.

[52] Carlos Ribeiro, Marco Tulio;singh, sameer; guestrin. Anchors: High-Precision Model-Agnostic Explanations Marco. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[53] Phillip E Pope, Soheil Kolouri, Mohammad Rostami, Charles E Martin, and Heiko Hoffmann. Explainability methods for graph convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10772–10781, 2019.

[54] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *Proceedings of the International Conference on Learning Representations Workshop*, 2014.

[55] Jianming Zhang, Sarah Adel Bargal, Zhe Lin, Jonathan Brandt, Xiaohui Shen, and Stan Sclaroff. Top-down neural attention by excitation backprop. *International Journal of Computer Vision*, 126(10):1084–1102, 2018.

[56] Hao Yuan, Jiliang Tang, and Shuiwang Ji. Xgnn: Towards model-level explanations of graph neural networks. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2020.

[57] Denny Zhou, Olivier Bousquet, Thomas N Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. In *Proceedings of the Conference on Neural Information Processing Systems*, 2004.

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

[59] Lekui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. Dynamic network embedding by modeling triadic closure process. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[60] Chenyi Zhuang and Qiang Ma. Dual Graph Convolutional Networks for Graph-Based Semi-Supervised Classification. In *the Web Conference*, 2018.

[61] James Atwood and Don Towsley. Diffusion-Convolutional Neural Networks. In *the Conference on Neural Information Processing Systems*, 2016.

[62] Di Jin, Ziyang Liu, Weihao Li, Dongxiao He, and Weixiong Zhang. Graph Convolutional Networks Meet Markov Random Fields : Semi-Supervised Community Detection in Attribute Networks. In *the AAAI Conference on Artificial Intelligence*, 2019.

[63] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the Conference on Neural Information Processing Systems*, pages 5998–6008, 2017.

[64] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2017.

[65] Martin G Everett and Stephen P Borgatti. Regular equivalence: General theory. *Journal of Mathematical Sociology*, 19(1):29–52, 1994.

[66] Francois Lorrain and Harrison C White. Structural equivalence of individuals in social networks. *The Journal of Mathematical Sociology*, 1(1):49–80, 1971.

[67] Douglas R White and Karl P Reitz. Graph and semigroup homomorphisms on networks of relations. *Social Networks*, 5(2):193–234, 1983.

[68] Martin G Everett, John P Boyd, and Stephen P Borgatti. Ego-centered and local roles: A graph theoretic approach. *Journal of Mathematical Sociology*, 15(3-4):163–172, 1990.

[69] Ryan A Rossi and Nesreen K Ahmed. Role discovery in networks. *IEEE Transactions on Knowledge and Data Engineering*, 27(4):1112–1131, 2015.

[70] Aliaksei Sandryhaila and José MF Moura. Discrete signal processing on graphs. *IEEE Transactions on Signal Processing*, 61(7):1644–1656, 2013.

[71] Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. In *Proceedings of the International Conference on Learning Representations*, 2018.

[72] Wayne W Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33(4):452–473, 1977.

[73] Aaron Clauset, Mark EJ Newman, and Cristopher Moore. Finding community structure in very large networks. *Physical Review E*, 70(6):066111, 2004.

[74] Sami Abu-El-Haija, Amol Kapoor, Bryan Perozzi, and Joonseok Lee. N-gcn: Multi-scale graph convolution for semi-supervised node classification. In *Proceedings of The Conference on Uncertainty in Artificial Intelligence*, 2019.

[75] Qing Lu and Lise Getoor. Link-based classification. In *Proceedings of the International Conference on Machine Learning*, 2003.

[76] David Jensen, Jennifer Neville, and Brian Gallagher. Why collective inference improves relational classification. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004.

[77] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1225–1234, 2016.

[78] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *Proceedings of the ACM International Conference on Web Search and Data Mining*, pages 459–467, 2018.

[79] Qu anyu Dai, Qiang Li, Jian Tang, and Dan Wang. Adversarial Network Embedding. In *the AAAI Conference on Artificial Intelligence*, 2018.

[80] Shirui Pan, Jia Wu, Xingquan Zhu, Chengqi Zhang, and Yang Wang. Tri-Party Deep Network Representation. In *the International Joint Conferences on Artificial Intelligence*, pages 1895–1901, 2016.

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

[82] Umang Sharan and Jennifer Neville. Temporal-relational classifiers for prediction in evolving domains. In *Proceedings of the IEEE International Conference on Data Mining*, 2008.

[83] Hogun Park, John Moore, and Jennifer Neville. Deep dynamic relational classifiers: Exploiting dynamic neighborhoods in complex networks. In *Proceedings of the International Conference on Web Search and Data Mining Workshop*, 2017.

[84] Ziwei Zhang, Peng Cui, Jian Pei, Xiao Wang, and Wenwu Zhu. Timers: Error-bounded svd restart on dynamic networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[85] Dingyuan Zhu, Peng Cui, Ziwei Zhang, Jian Pei, and Wenwu Zhu. High-order proximity preserved embedding for dynamic networks. *IEEE Transactions on Knowledge and Data Engineering*, 30(11):2134–2144, 2018.

[86] Jianxin Ma, Peng Cui, and Wenwu Zhu. Depthlgp: learning embeddings of out-of-sample nodes in dynamic networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[87] Lizi Liao, Xiangnan He, Hanwang Zhang, and Tat-Seng Chua. Attributed social network embedding. *IEEE Transactions on Knowledge and Data Engineering*, 30(12):2257–2270, 2018.

[88] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations*, 2014.

[89] Tianbao Yang, Yun Chi, Shenghuo Zhu, Yihong Gong, and Rong Jin. Detecting communities and their evolutions in dynamic social networks – a bayesian approach. *Machine Learning*, 82(2):157–189, 2011.

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

[91] Ryan L Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. In *Proceedings of the International Conference on Learning Representations*, 2019.

[92] Akshay Java, Xiaodan Song, Tim Finin, and Belle Tseng. Why we twitter: understanding microblogging usage and communities. In *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis*, pages 56–65, 2007.

[93] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[94] Joseph C Dunn. Well-separated clusters and optimal fuzzy partitions. *Journal of Cybernetics*, 4(1):95–104, 1974.

[95] Yehuda Koren. On spectral graph drawing. In *International Computing and Combinatorics Conference*, pages 496–508. Springer, 2003.

[96] Donglin Niu, Jennifer Dy, and Michael I Jordan. Dimensionality reduction for spectral clustering. In *Proceedings of International Conference on Artificial Intelligence and Statistics*, pages 552–560, 2011.

[97] Yang Wang, Zengru Di, and Ying Fan. Identifying and characterizing nodes important to community structure using the spectrum of the graph. *PloS one*, 6(11):e27418, 2011.

[98] Pablo Jensen, Matteo Morini, Márton Karsai, Tommaso Venturini, Alessandro Vespignani, Mathieu Jacomy, Jean-Philippe Cointet, Pierre Mercklé, and Eric Fleury. Detecting global bridges in networks. *Journal of Complex Networks*, 4(3), 2015.

[99] Zakariya Ghalmane, Mohammed El Hassouni, and Hocine Cherifi. Immunization of networks with non-overlapping community structure. *Social Network Analysis and Mining*, 9(1):45, 2019.

[100] István A. Kovács, Robin Palotai, Máté S. Szalay, and Peter Csermely. Community landscapes: An integrative approach to determine overlapping network module hierarchy, identify key nodes and predict network dynamics. *PLOS ONE*, 5(9), 09 2010.

[101] Jing-Dong J Han, Nicolas Bertin, Tong Hao, Debra S Goldberg, Gabriel F Berriz, Lan V Zhang, Denis Dupuy, Albertha JM Walhout, Michael E Cusick, Frederick P Roth, et al. Evidence for dynamically organized modularity in the yeast protein–protein interaction network. *Nature*, 430(6995):88, 2004.

[102] Nizar N Batada, Teresa Reguly, Ashton Breitkreutz, Lorrie Boucher, Bobby-Joe Breitkreutz, Laurence D Hurst, and Mike Tyers. Stratus not altocumulus: a new view of the yeast protein interaction network. *PLoS Biology*, 4(10), 2006.

[103] C Radhakrishna Rao. Diversity and dissimilarity coefficients: a unified approach. *Theoretical Population Biology*, 21(1), 1982.

[104] Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. Benchmark graphs for testing community detection algorithms. *Physical Review E*, 78(4):046110, 2008.

[105] Dan Garber, Elad Hazan, Chi Jin, Sham M Kakade, Cameron Musco, Praneeth Netrapalli, and Aaron Sidford. Faster eigenvector computation via shift-and-invert preconditioning. In *Proceedings of the International Conference on Machine Learning*, 2016.

[106] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the Conference on Neural Information Processing Systems*, 2013.

[107] Xin Rong. word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*, 2014.

[108] Matt Mahoney. *Large text compression benchmark*, 2011.

[109] Reza Zafarani and Huan Liu. *Social computing data repository*, 2009.

[110] Jana Diesner and Kathleen M Carley. Exploration of communication networks from the enron email corpus. In *Proceedings of the SIAM International Conference on Data Mining Workshop*, 2005.

[111] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *Proceedings of the The International Conference on Learning Representations*, 2020.

[112] Maximillian Nickel and Douwe Kiela. Poincaré embeddings for learning hierarchical representations. In *Proceedings of the Conference on Neural Information Processing Systems*, pages 6338–6347, 2017.

[113] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[114] Ling Huang, Donghui Yan, Nina Taft, and Michael I Jordan. Spectral clustering with perturbed data. In *Proc. of NeurIPS*, pages 705–712, 2009.

A APPENDIX

Lemma 1. Given a graph G, its set of clusters $C = \{C_1, ..., C_k\}$, and an arbitrary node $v_b \in V$, let G'_{-v_b} = Perturb(G, v_b) from Definition 5. If $v_b \in V$ has at least one inter-cluster edge, then $N_{asso}(C_i|G) \leq N_{asso}(C_i|G'_{-v_b})$ for $\forall C_i \subset C$.

Proof. By definition, $N_{asso}(C_i|G) = \frac{assoc(C_i,C_i|G)}{assoc(C_i,V|G)}$. After the degree-preserving perturbation, the denominator of $assoc(C_i, V|G)$ is unchanged, and the numerator $assoc(C_i, C_i|G)$, which considers edges only among C_i, increases or stays the same. We note that the increase happening to C_i includes v_b or v_q. □

Lemma 2. Let V be the nodes in graph G with clusters C, and G'_{-v_j} = Perturb(G, v_j) using Def. 5 with the perturbation ratio $\alpha = 1.0$ If we define

$$v_b = \arg\max_{v_j \in V} (N_{asso}(C_1, ..., C_k|G'_{-v_j}) - N_{asso}(C_1, ..., C_k|G)),$$

then v_b is the node with largest *Bridgeness*.

Proof. By definition, $N_{asso}(C_i|G) = \frac{assoc(C_i,C_i|G)}{assoc(C_i,V|G)}$. As in the proof of Lemma 1, when $v_i \in C_i$ has inter-cluster edges and the v_i is perturbed, $N_{asso}(C_i|G)$ increases. When $\alpha = 1.0$, the increase is from $\Sigma_{v_q \in \mathcal{N}_{inter}(v_b,C)} A[b, q]$, where $\mathcal{N}_{inter}(v_b, C)$ is a set of inter-cluster neighbors of v_b. Therefore, v_b who has the highest *Bridgeness* (thus, the highest inter-cluster degrees) increase at most for $N_{asso}(C_1, ..., C_k|G'_{-v_j})$. □

Theorem 1. Let V be a node set in G, and we have k clusters $\{C_1, ..., C_k\}$ in G. $G'_{-v_j} = \text{Perturb}(G, v_j)$ using Def. 5 with the perturb_ratio $\alpha = 1.0$, and eigenvectors $U \in \mathcal{R}^{|V|\times k}$ and $U'_{-v_j} \in \mathcal{R}^{|V|\times k}$ are given from G and G'_{-v_j}. Then, we have

$$\arg\max_{v_j \in V} Importance(v_j | U, U'_{-v_j}) = \arg\max_{v_j \in V} Bridgeness(v_j | C, G).$$

Proof. We note that $L = D - A$ and $L_{sym} = D^{-1/2} L D^{-1/2}$ and $A_{sym} = I - L_{sym}$. According to the Proposition 3 of [113],

$$\sum_{m=1}^{k} U_{:,m}^{T} L_{sym} U_{:,m} \approx \frac{1}{2} \sum_{v_i, v_j \in V} \left\| \frac{U_{i,1:k}}{\sqrt{d_i}} - \frac{U_{j,1:k}}{\sqrt{d_j}} \right\|^2 . \tag{A.1}$$

Using the proposition above, $U_{:,m}^{T} A_{sym} U_{:,m}$ could be decomposed as:

$$\begin{aligned}
&\sum_{m=1}^{k} U_{:,m}^{T} A_{sym} U_{:,m} = \sum_{m=1}^{k} U_{:,m}^{T} (1 - L_{sym}) U_{:,m} \\
&= \sum_{m=1}^{k} (U_{:,m}^{T} U_{:,m} - U_{:,m}^{T} L_{sym} U_{:,m}) = \sum_{m=1}^{k} (I - U_{:,m}^{T} L_{sym} U_{:,m}) \\
&\approx \sum_{m=1}^{k} (I) - \frac{1}{2} \sum_{v_i, v_j \in V} \left\| \frac{U_{i,1:k}}{\sqrt{d_i}} - \frac{U_{j,1:k}}{\sqrt{d_j}} \right\|^2 \text{ by Equation (A.1)} \\
&= \sum_{m=1}^{k} (I) - \frac{1}{2} DNPD(U) \approx N_{asso}(C_1, ..., C_k | G).
\end{aligned}$$

The last line above is from the objective function of spectral clustering, which finds U for maximizing $\sum_{m=1}^{k} U_{:,m}^{T} A_{sym} U_{:,m}$ s.t. $U^T U = I$, and it is equivalent to maximizing $N_{asso}(C_1, ..., C_k | G)$ [96]. Now we expect that N_{asso} and DNPD move in the same way. From Lemma 2, the difference in N_{asso} under perturbation is maximized by v_b who has the highest bridgeness. Therefore, the difference in DNPD under perturbation is also maximized by the same v_b so it can have the highest importance.

$\square$

Theorem 2. Let V be a node set in G, and there are k clusters $\{C_1, ..., C_k\}$. $G'_{-v_j} = \text{Perturb}(G,\ v_j)$ using Def. 5 with $\alpha = 1.0$, and eigenvectors $\hat{U} \in \mathcal{R}^{|V|\times k}$ and $\hat{U}'_{-v_j} \in \mathcal{R}^{|V|\times k}$ from DeepWalk embeddings are given from G and G'_{-v_j}. Then,

$$\arg\max_{v_j \in V} Importance(v_j|\hat{U}, \hat{U}'_{-v_j}) = \arg\max_{v_j \in V} Bridgeness(v_j|C, G).$$

Proof. In [78], the DeepWalk could be approximated by top-k largest singular values/vectors of $M = \frac{vol(A)}{T \cdot b} S$ where $S = (\sum_{r=1}^{T} P^r) D^{-1}$ and $P = D^{-1}A$. $\hat{U}$ is gotten from M. We note that the quality of the factorization under perturbation is closely related to the change of eigenvectors [114]. In the factorization, S could be decomposed as below:

$$\begin{aligned} S &= \left(\sum\nolimits_{r=1}^{T} P^r\right) D^{-1} = \left(\sum\nolimits_{r=1}^{T} (D^{-1}A)^r\right) D^{-1} \\ &= D^{-1/2} \sum\nolimits_{r=1}^{T} (D^{-1/2}AD^{-1/2})^r D^{-1/2} \\ &= D^{-1/2} \sum\nolimits_{r=1}^{T} (A_{sym})^r D^{-1/2}. \end{aligned} \tag{A.2}$$

Meanwhile, we have $A_{sym} = U\Lambda U^T$ to get U and find clusters in spectral clustering. From this, we can get $\sum_{r=1}^{T}(A_{sym})^r = U(\sum_{r=1}^{T} \Lambda^r)U^T$. Because the additional normalization in S by $D^{-1/2}$ is not likely to change the final cluster IDs much, we have $\sum_{m=1}^{k} \hat{U}_{:,m}^T S \hat{U}_{:,m} \approx N_{asso}(C_1, ..., C_k|G)$, as in Theorem 1. Therefore, the difference in its DNPD under perturbation is also maximized by v_b who has the highest bridgeness and the perturbation from v_b has the largest difference in DNPD. Thus, v_b has the highest *Importance* value in the eigenvector of DeepWalk. □

Lemma 3. Let G be a graph with a set of clusters C and associated embedding W. Let $v_b \in C_b$ be a bridge node and $v_c \in C_b$ be a node with the same degree but fewer inter-community edges (thus, lower *bridgeness*) than v_b. Suppose that W is a "good embedding" as in Definition 7. Then if GRAPH-GD considers all edges without sampling, GRAPH-GD(v_b) > GRAPH-GD(v_c).

Proof. We can decompose GRAPH-GD(v_b) using cluster memberships as below:

$$\text{GRAPH-GD}(v_b) = \text{MEAN}(\{d(\vec{w}_i, \vec{\hat{w}}_i|\vec{w}_b), v_i \in \mathcal{N}(v_b))\})$$
$$= \text{MEAN}(\{d(\vec{v}_i, \vec{\hat{w}}_i|\vec{w}_b), v_i \in \mathcal{N}(v_b) \cap C_b\}+$$
$$\{d(\vec{w}_k, \vec{\hat{w}}_k|\vec{w}_b), v_k \in \mathcal{N}(v_b) \cap C/C_b\})$$

Meanwhile, GRAPH-GD(v_c) is also described as:

$$\text{GRAPH-GD}(v_c) = \text{MEAN}(\{d(\vec{w}_i, \vec{\hat{w}}_i|\vec{w}_c), v_i \in \mathcal{N}(v_c))\})$$
$$= \text{MEAN}(\{d(\vec{w}_i, \vec{\hat{w}}_i|\vec{w}_c), v_i \in \mathcal{N}(v_c) \cap C_b\}+$$
$$\{d(\vec{w}_k, \vec{\hat{w}}_k|\vec{w}_c), v_k \in \mathcal{N}(v_c) \cap C/C_b\})$$

In equations above, $|\{\mathcal{N}(v_b) \cap C/C_b\}| > |\{\mathcal{N}(v_c) \cap C/C_b\}|$ by the assumption about inter-community edges. Using the definition 7 and the same degree constraint for v_b and v_c, we can see that GRAPH-GD(v_b) − GRAPH-GD(v_c) > 0. □

Lemma 4. Let v_b be a bridge node and B-Cluster(v_b) is a set of clusters that are bridged by v_b. Assume there is a cluster $C_i \subset$ B-Cluster(v_b) where the nodes in C_i are densely connected to each other. If v_i is a support node for giving v_b the bridging role for C_i, then $h(v_b, v_i) > 1$. If v_i is a not support node for C_i and v_b, then $h(v_b, v_i) = 1$.

Proof. As in Figure 5.2b, there exist a sum of gradients generated by pairs, $\{(v_i, v_j)|v_j \in C_i, v_j \in \mathcal{N}(v_i)\}$, and the gradient update by them becomes $\vec{w}_{C_i} - \vec{w}_i$ under the assumption of the dense edges among nodes in C_i. In other words, the update makes $\vec{w}_i$ move to the centroid of the cluster and we call it as $\vec{p}$. Meanwhile, when the gradient generated only by a pair (v_i, v_b) is $\vec{q}$, the final gradient update becomes $\vec{q} - \vec{p}$, which means that the gradient update from (v_i, v_b) is subtracted by the gradients from all pairs from v_i. Thus, the final update $\vec{q} - \vec{p} = \vec{w}_b - \vec{w}_{C_i}$. If v_i is a support node, then the angular distance between $\vec{w}_b - \vec{w}_i$ and $\vec{w}_b - \vec{w}_{C_i}$ is always between $-90°$ and $90°$. As a result, its cosine distance is larger than 0, and $h(v_b, v_i) > 1$. Similarity, If v_i is not a support node for C_i and v_b, its cosine distance is negative. Therefore, $h(v_b, v_i) = 1$. □

Lemma 5. Let v_b be a bridge node and B-Cluster(v_b) is a set of clusters that are bridged by v_b. Assume there is a cluster $C_i \subset$ B-Cluster(v_b) where the nodes in C_i are densely connected each other. If v_b is a bridge node and its context nodes include at least one support node, then GRAPH-wGD(v_b) > GRAPH-GD(v_b).

Proof.

$$\begin{aligned}&\text{GRAPH-wGD}(v_b) - \text{GRAPH-GD}(v_b)\\ &= \text{MEAN}\left\{\left|\frac{\partial \mathcal{L}(v_b, v_i)}{\partial \vec{w}_i}\right| \cdot (h(v_b, v_i) - 1), v_i \in \mathcal{N}(v_b)\right\}\\ &> 0 \text{ from Lemma 4.}\end{aligned}$$

Therefore, GRAPH-wGD(v_b) > GRAPH-GD(v_b). □

Lemma 6. When context nodes of a node $v_c \in V$ do not include support nodes, GRAPH-GD(v_c) = GRAPH-wGD(v_c).

Proof. As in Lemma 4, if there is no support node in v_c's context node set, $\{v_i \in \text{Context}(v_c)\}$, then $h(v_c, v_i) = 1$ for any v_i. Therefore, GRAPH-GD(v_c) = GRAPH-wGD(v_c). □

Theorem 3. Let G be a graph with a set of clusters C and associated embedding W. We denote $v_b \in C_b$ be a bridge node and $v_c \in C_b$ be a node with the same degree but fewer inter-community edges (thus, lower *bridgeness*) than v_b. Assume that the context nodes of v_b include at least one support node, and the context nodes of v_c do not include a support node. If W is a "good embeddings" (Def. 7), nodes in C_b are densely connected with each other, and GRAPH-wGD and GRAPH-GD consider all edges without sampling, then GRAPH-wGD(v_b) > GRAPH-GD(v_b) > GRAPH-GD(v_c) = GRAPH-wGD(v_c).

Proof. Using Lemma 5 and Lemma 3, we can see that GRAPH-wGD(v_b) > GRAPH-GD(v_b) and GRAPH-GD(v_b) > GRAPH-GD(v_c), respectively. In addition, Lemma 6 shows that GRAPH-wGD(v_c) = GRAPH-wGD(v_c) under the assumption that context nodes of v_c do not include support nodes. As a result, we can prove that GRAPH-wGD(v_b) $>$ GRAPH-GD(v_b) $>$ GRAPH-GD(v_c) $=$ GRAPH-wGD(v_c). $\square$

VITA

Hogun Park received his Bachelor of Science from Korea Advanced Institute of Science and Technology (KAIST) in 2006, followed by his Master of Science from KAIST in 2008. Hogun then moved to the computer science department at Purdue University to pursue his PhD under Professor Jennifer Neville. During his graduate study, he has tackled semi-supervised machine learning problems in relational data (e.g., social network and behavioral data) and natural language processing (NLP) problems. He has published a number of works at various conferences/journals and has filed several US/KR patents. He also has 6 years of industry/national lab experience including 9 months of internship with IBM Research and built data analysis pipelines to manage nationwide travel information and event-related tweets at Korea Institute of Science and Technology (KIST).