Designing better Graph Convolutional Networks: Scaling Graph Propagation based Neural Networks for Collective Classification

A THESIS

submitted by

PRIYESH V

for the award of the degree

of

MASTER OF SCIENCE

(by Research)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

JUNE 2019

THESIS CERTIFICATE

This is to certify that the thesis titled **Designing better Graph Convolutional Networks: Scaling Graph Propagation Neural Networks for Collective Classification**, submitted by **Priyesh V**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Science (by Research)**, is a bona fide record of the research work done by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Balaraman Ravindran
Research Guide
Professor
Dept. of CSE
IIT-Madras, 600 036

Place: Chennai

Date: 27-06-2019

ACKNOWLEDGEMENTS

Herein, I wish to express my sincere gratitude to everyone who has and is continuing to influence me positively. The first and foremost influential set of people in my life are my parents, Vijayan and Muthulekshmi and my research advisor, Prof. Ravindran. They have believed in me and supported me in multiple aspects of my life. Especially Ravi sir, who gave me my freedom to explore in my research and entrepreneurial journey.

I have closely known Ravi sir for more than five years now dating back from my days at Ericsson. Ravi sir has been my dose of inspiration at IIT Madras. I have mentioned multiple times to my friends that he is our dark knight. The hero who works tirelessly to provide a better research stage to his students and his country. Seeing him push his limits every day allows me to give my best too. I enjoyed every interaction with him on research, especially our brainstorming sessions on numerous projects. I learned from him that it is essential to work on a good problem and also take pride in it.

Secondly, I would like to thank Prof. Mitesh. He has always made me feel comfortable with him just like friends do. I'm thankful to him for giving me the opportunity to work with him and explore NLP without any background. He has made realize the importance of meticulous preparation and doing disciplined research.

I'm privileged to work with Prof. Srinivasan. Our interactions improved my understanding of numerous problems in Social networks and data mining. I'm grateful to him for honing my analysis skills from our discussions.

A special thanks to Yash Chandak and Anasua. Numerous night outs and endless discussions with them has resulted in this thesis. Both of them were instrumental in finishing this thesis.

I'm grateful to be a part of RISE lab and RBC-DSAI. They have provided me with the opportunity to work and interact with many brilliant minds. Beethika, Preksha, Patanjali, Nikita, Sapana, Mittal, Sudarshan, Maurya, Tarun, Thiagarajan, Azhagesan, Shubho, Sidharth and Pallavi are my cool peers whom I mostly hang out with and do

research in the lab. Without them, I wouldn't have had the opportunity to be aware, learn and work on diverse and exciting projects. Course projects wouldn't have been any fun without my awesome teammates, Vishruit, Jyotsna and Suresh.

I also thank my close friends, Suresh, Madhuri and Prasanna who made stay comfortable at IIT Madras. Even after they left IIT Madras, they have always stayed in touch and motivated me. Mentoring and working with juniors were super fun. I thank Yash, Aakash, Sachin, Harsha, Ujjuwal, Mohan, Karthik, Karan, Gautham, Dhruv, Sowmya, Charumathi, Sankaran, Kavita, PavitraKumar, Aditya, Vijay, Heeth, Vinodhini, Vasumathi, Arun, Divakar, Arjhun and the school kids who came for the RSIC internship for providing me a chance to learn and grow with you.

Finally, I would like to end with thanking my most favorite set of people. I thank (i) My brother, Ratheesh for being supportive always (ii) my best buddies who are my constants and go to persons throughout my life: Ram, Praveen, Vimal, Aswini, Raga, Keerthana, Nandhini, Aswathy, Prakash, Gautham, Manoj and Praveen and (iii) my mentors Shivashankar and Sarath.

ABSTRACT

KEYWORDS: Semi-Supervised Learning, Representation Learning, Relational Learning, Collective Classification, Node Classification, Deep Learning, Graph Convolutional Networks

In many real-life applications, entities in an environment are not independent but rather influenced by each other through their interactions. Such relational datasets are popularly modeled as graphs where the entities make up the node, and the edges represent an interaction. Learning algorithms for node classification on such relational data is different from conventional machine learning algorithms which ignore the relational information and assumes samples (entities) are drawn from an IID. Relational learning algorithms for node classification should adhere to the structure of the data and capture complex correlations between the node and its neighbors for the classification task.

Given a graph where every node has specific attributes associated with it, and some nodes have labels associated with them. Collective Classification is the task of assigning labels to every unlabeled node using information from the node as well as its neighbors. It is often the case that a node is not only influenced by its immediate neighbors but also by it's higher order neighbors, multiple hops away. Variants of Graph Convolutional Neural Networks (GCNs) are the state-of-the-art neural network architectures for Collective Classification. GCNs are end-to-end differentiable variations of recursively defined graph kernels that aggregate and filter multi-hop neighborhood information.

In this work, we propose a Higher Order Propagation Framework, HOPF, which provides an iterative inference mechanism for these powerful differentiable kernels. Such a combination of classical iterative inference mechanism with recent differentiable kernels allows the framework to learn graph convolutional filters that simultaneously exploit the attribute and label information available in the neighborhood. Further, these iterative differentiable kernels can scale to larger hops beyond the memory limitations of existing differentiable kernels. The proposed modular framework enables us to better analyze the relative merits and demerits of numerous existing models as specific

instantiations of our framework. Through these analyses, we observe two hitherto incapacities of current models that results in Node Information Morphing and Recursive weight dependency issues.

The Node Information Morphing issue points out that the original information at the node is morphed or averaged out by its multi-hop neighborhood information exponentially. To alleviate this issue, we propose an instantiation of HOPF called the Node Information Preserving (NIP) kernels which addresses the NIM issue by explicitly preserving the node information at every propagation step of GCN. Secondly, we focus on the problems imposed by Recursive weight dependencies in GCNs. These recursively defined networks have limited representation capacity to regulate information from multiple hops independently. We propose a simple extension to GCNs, Fusion Graph Convolutional Networks that contains a linear fusion component that mitigates this issue by leveraging our observation that GCNs are graph kernels with binomial basis. We extensively evaluate and demonstrate the superiority of the proposed models on datasets from biology, bibliography, sociology and finance domains under transductive and inductive learning setup.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

SSL	Semi-Supervised Learning
NRL	Network Representation Learning
CC	Collective Classification
PSD	Positive Semi-Definite
IID	Independent and Identically distributed
PCA	Principal Component Analysis
NMI	Normalized Mutual Information
KKT	Karush−KuhnTucker
ReLU	Rectified Linear Unit
NMF	Non-negative Matrix Factorization
SS-NMF	Semi-Supervised NMF
GRU	Gated Recurrent Units
LSTM	Long Short Term Memory
GNN	Graph Neural Networks
GCN	Graph Convolutional Neural network
F-GCN	Fusion GCN
GraphSAGE	Graph SAmple and Aggregate
GS-MEAN	GraphSAGE with MEAN pooling aggregator
GS-MAX	GraphSAGE with MAX pooling aggregator
GS-LSTM	GraphSAGE with LSTM aggregator
HOPF	Higher Order Propagation Framework
NIM	Node Information Morphing
NIP	Node Information Preserving
I-NIP	Iterative NIP
WL	Weisfeiler Lehman
LP	Label Propagation
ICA	Iteartive Collective classification Algorithm
t-SNE	t-Distributed Stochastic Neighbor Embedding

GEMSEC	Graph Embedding with Self Clustering
ComE	Community Embedding
MMDW	Max-Margin DeepWalk
GPU	Graphic Processing Unit
CONCAT	Concatenate
PPI	Protein-Protein Interaction
BL_NODE	Baseline: Node feature
BL_NEIGH	Baseline: Neighbor features

NOTATION

$\mathbb{R}$	Set of real numbers
G	Graph
V	Vertex set of a graph, G
E	Edge set of a graph, G
n	Number of nodes, $\|V\|$ in a graph G
f	Number of features for nodes, V
S	Set of labeled nodes
U	Set of unlabeled nodes
X	Nodes' feature matrix; $X \in \mathbb{R}^{n*f}$
l	Number of labels for nodes, V
Y	Nodes' label matrix; $Y \in [0,1]^{\|S\|*f}$
$\hat{Y}$	Nodes' predicted label matrix; $Y \in \mathbb{R}^{n*f}$
I	Identity matrix $\in \mathbb{R}^{n*n}$
A	Adjacency matrix of a graph, G; $A \in \mathbb{R}^{n*n}$
P	Proximity matrix of a graph, G; $P \in \mathbb{R}^{n*n}$
D	Diagonal Degree matrix of a graph, G; $D \in \mathbb{R}^{n*n}$
L	Laplacian matrix of a graph, G; $L \in \mathbb{R}^{n*n}$
$\hat{L}$	Re-normalized Laplacian matrix of a graph, G; $L \in \mathbb{R}^{n*n}$
K	Desired neighborhood size in hops; $K \geq 0$
C	Number of convolutional layers in GCNs); $C \leq K$
	Neighborhood size captured by GCNs; Typically, C=K
k	k^{th} differentiable layer of GCNs; $k \leq K$
T	Number of iterative learning or inference steps
t	t^{th} iterative step; $t \leq T$
h_k	k^{th} hidden layer of a neural network
σ_k	k^{th} neural network layer's activation function
Φ_k	Node features of k^{th} layer of GCNs
Ψ_k	Neighborhood features of k^{th} layer of GCNS
Ξ_t	Neighborhood features of t^{th} iterative step
$\hat{Y}_t$	Label based neighborhood features of t^{th} iterative step
W_k	Weights of k^{th} layer neural network; $W_k \in \mathbb{R}^{d*d}$
W_L	Weights of the label layer of a neural network;
W_k^{Φ}	Weights associated with k^{th} layer's node features, Φ_k
W_k^{Ψ}	Weights associated with k^{th} layer's neighborhood features, Ψ_k
α	Importance of nodes features, Φ_k
β	Importance of neighborhood features , Ψ_k
$F(A)$	Function of Adjacency matrix
O	Big O notation

CHAPTER 1

Introduction

An efficient representation of data is critical for analysis and learning tasks. Efficient in terms of space allows data processing algorithms to scale and efficient in terms of information allows processing algorithms to effectively uncover the different explanatory factors behind the variation in data. Devising learning algorithms explicitly to derive efficient representations for the end applications is crucial as it allows to conveniently encode general priors in the data like smoothness, sparsity, clustering, manifolds, multiple explanatory factors, etc.

Manifold learning and the more generic Representation learning are the two subfields of machine learning that focus on finding efficient data representations for machine learning tasks with an intent to uncover the structure in the data. Manifold learning methods are dimensionality reduction techniques that find a dense new coordinate system in low-dimensional regions of the original data, whereas Representation learning methods can also produce sparse over-complete features in a high dimensional space with more dimensions than the input (Poultney *et al.*, 2007). Matrix (Tensor) factorization and Neural networks are two powerful and commonly used representation learning methods. In the current era of the burgeoning field of Deep Learning, deep neural network architectures have become the unanimous choice of representation learning in a majority of the domains for large data.

1.1 Network Representation learning

In many real-life applications such as social networks, citation networks, protein interaction networks, etc., the entities in an environment are not independent but rather influenced by each other through their interactions. Network representation learning (NRL), focuses on extracting useful features in such complex networked data. Network data, unlike conventional data, does not follow the assumption that samples are drawn from an independent and identically distribution (IID) and additionally have some inherent

structure. Network data may contain homogeneous or heterogeneous nodes which may exhibit one or more (un)directed homophilous or heterophilous relationships between nodes or even both. Network data can also contain one or more contextual features associated with nodes or relations. Representation learning algorithms for networks should preserve the structure in the data explicitly and encode the necessary priors required for the end task. Depending on the task, representations are learned for nodes, relations and the entire graph as such.

Such network data or relational data have been popularly modeled as graphs where the entities make up the node, and the edges represent an interaction. The use of graph-based learning algorithms has increasingly gained traction owing to their ability to model structured data. Categorizing such entities requires extracting relational information from their multi-hop neighborhoods and combining that efficiently with their features. Summarizing information from multiple hops is useful in many applications where there exists semantics in local and group level interactions among entities. Thus, defining and finding the significance of neighborhood information over multiple hops becomes an essential aspect of the problem.

1.2 Collective Classification

Given a graph where every node has specific attributes associated with it, and some nodes have labels associated with them, Collective Classification (CC) (Neville and Jensen (2000); Lu and Getoor (2003); Sen P *et al.* (2008)) is the task of assigning labels to the remaining unlabeled nodes in the graph. A vital task here is to extract relational features for every node which consider not only the attributes of the node but also the attributes and labels of its partially labeled neighborhood. The Collective Classification task is a classical Semi-Supervised Learning (SSL) problem where node representations are obtained by leveraging the information from large amounts of unlabeled nodes in addition to the information from the labeled nodes. It is often the case that a node is not only influenced by its immediate neighbors but also by it is higher order neighbors, multiple hops away which may be both labeled and unlabeled.

Traditionally handcrafted features were widely used to capture relational information. Popular methods used mix of count statistics of label distribution (Neville and

Jensen (2003); Lu and Getoor (2003)), relational properties of nodes like degree, centrality scores (Gallagher and Eliassi-Rad (2010)), and attribute summaries of immediate neighborhood, etc. Limited by manual engineering, these traditional methods only used raw features built from information associated with immediate (first order or one hop) neighbors.

The recent surge in deep learning has shown promising results in extracting important semantic features and learning good representations for many machine learning tasks. Deep learning models for such relational node classification tasks can be broadly categorized into models that either learn node representation with structural regularization (Perozzi *et al.*, 2014; Wang *et al.*, 2016) or those that ignore explicit structural regularization and learn to aggregate neighbors' information (Hamilton *et al.*, 2017; Moore and Neville, 2017; Kipf and Welling, 2016). The former is limited to work only in networks that exhibit high homophily, as they enforce the representation of a node and its neighbor to be similar. The latter methods, on the other hand, do no make such assumptions.

Initial works of Frasconi *et al.* (1998) on relational feature extraction with neural nets primarily relied on recursive neural nets to process the graph data, limited by their ability to deal only with directed ordered acyclic graphs, Scarselli *et al.* (2009) and Gori *et al.* (2005) introduced Graph Neural Networks (GNNs) which used recursive neural nets to propagate information in any general graph iteratively. However, these GNNs were limited to problems where the entire graph can fit into memory. To extend the work to sequence generation problems on graphs, Li *et al.* (2015) adapted Graph Recurrent Units (GRUs) (Cho *et al.*, 2014) in the propagation step. Recently, Moore and Neville (2017) proposed a Long Short Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) based sequence embedding model for node classification but it required randomly ordering first hop neighbors' information, thus essentially discarded the topological structure.

To directly deal with the graph's topological structure, Bruna *et al.* (2013) defined convolutional operations in the spectral domain for graph classification tasks, but required computationally expensive eigendecomposition of the graph Laplacian. To reduce this requirement, Defferrard *et al.* (2016) approximated the higher order relational feature computation with first order Chebyshev polynomials defined on the graph

Laplacian. Graph Convolutional Networks (GCNs) (Kipf and Welling, 2016) adapted them to Semi-Supervised node level classification tasks. GCNs simplified Chebyshev Nets by recursively convolving one-hop neighborhood information with a symmetric graph Laplacian. Recently, Hamilton *et al.* (2017) proposed a generic framework called GraphSAGE with multiple neighborhood aggregator functions. GraphSAGE works with a partial (fixed number) neighborhood of nodes to scale to large graphs. GCN and GraphSAGE are the current state-of-the-art approaches for transductive and inductive node classification tasks in graphs with node features. These end-to-end differentiable methods provide impressive results besides being efficient regarding memory and computational requirements.

1.3 Contributions of this Thesis

In this thesis, we argue that though Graph propagation based Neural Networks such as the GCNs and GraphSAGE family are powerful and provide impressive state-of-the-art results, they still cannot efficiently obtain information from multiple hops. Multiple issues restrain these networks from extracting relevant multi-hop neighborhood information for the end tasks. The primary contribution of this thesis lies in identifying and analyzing these issues along with providing solutions to alleviate these issues. The contributions are presented in the following two works.

1.3.1 Higher Order Propagation Framework

Our main contributions presented in this work are:

- *A modular graph kernel that generalizes many existing methods*. Through this, we discuss a hitherto unobserved phenomenon which we refer to as *Node Information Morphing* (NIM). We discuss its implications on the limitations of existing methods and then discuss a novel family of kernels called the *Node Information Preserving (NIP) kernels* to address these limitations.
- A *hybrid Semi-Supervised Learning (SSL) framework* for higher order propagation that couples differentiable graph kernels with an iterative learning and inference procedure to aggregate neighborhood information over farther hops. This allows differentiable kernels to *exploit label information* and further *overcome excessive memory constraints* imposed by multi-hop information aggregation.
- An extensive *experimental study on eleven datasets* from different domains. We demonstrate the NIM issue and show that *the proposed iterative NIP model is robust* and overall outperforms existing models.

1.3.2 Fusion Graph Convolutional Networks

Our main contributions presented in this work are:

- We point out that current state-of-the-art graph convolutional nets lack the representation capacity to regulate different neighborhood information independently.
- We also show that these models capture K-hop neighborhood information by a K^{th} order binomial. We take advantage of this to build a binomials basis for the K-hop neighborhood space with outputs from different graph layers corresponding to different hop. With the binomial basis, we define a simple linear fusion layer that can capture any required combination of hops for the end task.
- We propose F-GCN, a simple extension to GCNs with the proposed fusion layer. We show that the proposed model outperforms the state-of-the-art models on nine out ten datasets while being highly competitive on the other.

1.4 Outline of the thesis

The remainder of this thesis is organized as follows. Chapter 2 provides the necessary background on graph-based Semi-Supervised learning with Laplacian regularizers and also provides brief literature on the current state-of-the-art neural network models for Semi-Supervised Collective Classification. In Chapter 3, we discuss the proposed Higher Order Propagation Framework, HOPF which provides a generic graph kernel with an iterative learning framework. We study a hitherto unobserved issue of Node Information Morphing with the aid of the generic kernel. The iterative learning framework enables scaling of Graph Convolutional Networks (GCNs) beyond their memory limitations. This chapter is concluded with an extensive experimental study on eleven datasets from numerous domains. In Chapter 4, we discuss the work on Fusion Graph Convolutional Networks, F-GCNs. We analyze the limitations on the representation capacity of GCNs to regulate information from multiple hops independently. We show that GCNs are differentiable graph kernels with binomial basis. Then, we leverage this observation to propose a simple fusion component that empowers existing GCNs representation capacity to capture efficient multi-hop information. This chapter is concluded with an extensive experimental study. Finally, in Chapter 5 we briefly summarize the implications of the contributions and also discuss possible future works.

CHAPTER 2

Background and Related works

Network data have inherent relational structure among the nodes of a graph. The relational structure (edges) among nodes might encode additional latent correlated information which might be useful. For example, social networks exhibit correlations such as homophily (individual characteristics affect social connections), influence (social connections influence individual characteristics) and community (social connections determine the social group). Such relational information is deemed useful if it gives additional evidence that might improve the belief obtained from using only the node features. Such benefit might arise when the node features are not sufficient or robust as a stand-alone information for the end task.

Representations for the graph are learned at node or edge or graph level depending on the end task. Herein, we focus on learning representations at node (vertex) level for the task of classifying nodes. Classifying nodes in Information networks such as social networks, citation networks, protein interaction networks, etc. involve characterizing the nodes' information along with its neighborhood information concerning their relational structure, attributes and their complex non-linear correlation with the labels.

In this section, we start by defining a Collective Classification (CC) dataset and other notations commonly used by different CC models in this work. Next, we introduce the CC task as a graph-based Semi-Supervised Learning (SSL) problem. Then, we discuss how spectral kernels, in general, are used as regularizers and feature extracting filters to preserve and leverage relational information. Having reviewed the background on learning with spectral kernels, we move on to discuss the current state-of-the-art spectral kernels based CC models, the Graph Convolutional Networks (GCNs) and their relation to Weisfeiler-Lehman (WL) algorithm. Following GCNs, we explain GraphSAGE, a GCN variant with more generic relational filters. Then, we also discuss the classical Iterative Collective Classification algorithm, which we later leverage in our proposed work to make GCNs more scalable and powerful feature extractors.

Additionally, in the end, we also provide details on our other related work for

leaning semi-supervised embeddings for non-attributed graphs, 'Semi-Supervised Non-Negative Matrix Factorization for Clusterable embeddings' (SS-NMF). This work was jointly done in Collaboration with Anasua Mitra, a PhD student at IIT Guwahati. We report the entire study of this work here in the related section as I have significantly contributed to this project and that the contributions of this work will appear in her thesis.

2.1 Notations

Let $G = (V, E)$ denote a graph with a set of vertices, V, and edges, $E \subseteq V \times V$. Let $|V| = n$. The set E is represented by an adjacency matrix $A \in \mathbb{R}^{n \times n}$ and let $D \in \mathbb{R}^{n \times n}$ denote the diagonal degree matrix defined as $D_{ii} = \sum_j A_{i,j}$.

A Collective Classification dataset defined on graph G comprises of a set of labeled nodes, S, a set of unlabeled nodes, U with $U = V - S$, a feature matrix: $X \in \mathbb{R}^{n \times f}$ and a label matrix: $Y \in \{0, 1\}^{|S| \times l}$, where f and l denote the number of features and labels, respecetively. Let $\hat{Y} \in \mathbb{R}^{n \times l}$ denote the predicted label matrix.

In this work, neural networks defined over K-hop neighborhoods have K aggregation or convolution layers with d dimensions each and whose outputs are denoted by $h_1, \ldots, h_K$. We denote the learnable weights associated with k-th layer as W_k^{ϕ} and $W_k^{\psi} \in \mathbb{R}^{d \times d}$. The weights of the input layer (W_1^{ϕ}, W_1^{ψ}) and output layer, W_L are in $\mathbb{R}^{f \times d}$ and $\mathbb{R}^{d \times l}$ respectively. Iterative inference steps are indexed by $t \in (1, T)$. Let σ_k be the activation function of the k^{th} layer in the neural network.

2.2 Graph-based Semi-Supervised learning

Semi-Supervised Learning (SSL) leverages unlabeled data to achieve improved generalization on supervised tasks. Typically, it is useful when the supervision is limited. In this work, we focus on the standard SSL setup where the supervision is provided in the form of target labels for data instances.

Semi-Supervised Learning aims at improving the performance of the supervised learning task by obtaining a better estimate of the underlying data distribution with the

available additional unlabeled data. Similar to the necessary assumptions that are made for supervised learning such as continuity, independent and identically distributed data (IID), etc., there are necessary assumptions required for SSL (Chapelle *et al.* (2009)) to work. It is vital to ensure that these assumptions, specifically those on the data, are satisfied when coupled with models that jointly learn representations for data. The popular graph-based SSL paradigm that is built on the manifold assumption preserves the geometry of data especially under the change of representation.

Graph-based SSL methods model the underlying data manifold as a graph and utilize it to enforce smoothness on a target space. The graph can either be computed from the data or given apriori. Both labeled and unlabeled data are represented as nodes, and the edges denote a notion of similarity. When the graph is not given apriori, the weights on the edges are computed with some similarity kernel on the input space. The required smoothness properties are encoded with an appropriate spectral (graph) kernel (Smola and Kondor (2003)).

2.3 Spectral Kernels

The different graph-based learning approaches for the Collective Classification task can be broadly classified into two paradigms based on the use of spectral kernels:

- Graph-based regularization with kernels as similarity functions.
- Graph feature extraction with kernels as filters.

In a node classification task, the kernels operate in the vertex domain, and hence the regularization and feature extraction are over the neighborhood of vertices. Regularization based methods enforce nodes be similar to their neighbors in the label or embedding space, and feature extraction methods aggregate neighborhood information and combine it with the node features.

2.3.1 Regularization with Spectral kernels

Let L denote the normalized symmetric Laplacian operator, where $L = I - D^{-1/2}AD^{-1/2}$ with I being the identity matrix. The Laplacian operators defined on undirected graphs

are symmetric matrices. This property of Laplacian matrices is the foundation for the spectral theory. As graph Laplacian L is a real symmetric matrix, it can be diagonalized to obtain orthonormal eigenvectors U with associated real and non-negative eigenvalues Λ. The multiplicity of eigenvalue 0, corresponds to the number of connected components in the graph. The eigenvectors corresponding to smaller eigenvalues of L encode smooth variations for connected nodes i.e $|U_i(m) - U_i(n)|$ is small.

The Laplacian operator on a function of graph signal, $f = f(X)$ penalizes the change between adjacent vertices as described in 2.1. Following from the definition of a Positive Semi-Definite (PSD) matrix, the graph Laplacian operator can be used as a semi-norm on signals making it a useful tool for regularization.

$$< f, Lf >= f^T Lf = \sum_{(i,j)\in E} ||f_i - f_j||^2 \geq 0 \tag{2.1}$$

We can define a class of regularization functionals, $r(\theta)$ on the graph by parameterizing a function over the the Laplacian which would be a function over the eigen values as given in Eqn: 2.3. The choice of function should be driven with the thought that we need to penalize V_i which has large Λ_i, hence the r_Λ should monotonically increase with Λ.

$$L = \quad U^T \Lambda U \tag{2.2}$$

$$r(L) = \quad r(\Lambda) = \quad U^T r(\Lambda) U \tag{2.3}$$

The most common choices of graph regularizers, r are the plain Laplacian 2.4 and the regularized Laplacian 2.5 with $\sigma^=1$ followed by 2.6.

$$r(\Lambda) = I\Lambda \tag{2.4}$$

$$r(\Lambda) = (I + \sigma^2 \Lambda) \tag{2.5}$$

$$r(\Lambda) = exp(\sigma^2/2L) \tag{2.6}$$

As long as $r(L)$ is a PSD matrix, we can define a Reproducing Hilbert space with kernel, $\kappa = r(L)^{-1}$. For more on graph regularization and kernels, refer (Smola and Kondor (2003)).

Laplacian smoothing

These Regularization kernels are used to encode smoothening priors based on the relational structure of G. Laplacian regularizers have been extensively used for semi-supervised node classification. Below, we provide a functional definition of Laplacian regularizer parameterized by the graph of consideration and the space to be smoothed.

$$LS(P, f(X)) = \sum_{(i,j)\in E} \|f(X_i) - f(X_j)\|^2 = \quad f^T \Delta(P) f \tag{2.7}$$

Let Δ denote the graph Laplace operator. Let $LS(P, f(X))$ denote Laplacian Smoothing (LS) on the target space, $f(X)$ with $L = \Delta(P)$, where $P \in \mathbb{R}^{n*n}$ is some defined proximity matrix. As mentioned earlier, Laplacian smoothing can be applied on the label space $f(X) = L$, data projections $X or f(X)$ or on a latent cluster space, $f(X) = C$ as proposed in this work.

$$Loss = \quad Loss_{Supervised} + \lambda LS(P, f(X)) \tag{2.8}$$

The regularization term is explicitly added to learning objective and is optimized along with the supervised loss as in 2.8. The λ in the equation is a trade-off parameter which weighs neighborhood similarity with the node features. λ is chosen by cross-validation to determine the importance of node features. Laplacian SVM (Belkin *et al.* (2006)) is a popular model which couples a Laplacian regularizer to the Hinge loss and one can also view Label propagation (Zhu and Ghahramani (2002)) as Laplacian smoothening on the label space with a k-nearest neighbor classifier.

2.3.2 Convolutions with Spectral kernels

The spectral of the Laplacian help us form analogous theory to Fourier transforms leading to Graph Fourier Transforms enabling us to model filters which can be used for convolution operation in the original graph space as in Eqn: 2.9. Unlike kernels for regularization which focused on monotonic scaling of the eigen values, kernels for filtering

are chosen to parameterize the combination of the laplacian basis.

$$g_\theta * x \quad = \quad U g_\theta(\Lambda) U^T x \tag{2.9}$$

The non-parametric kernels as in Eqn: 2.10 are potent alternatives to parametric filters as they are not biased to any specific family of spectral transformations and also have higher degrees of freedom of the order of number of nodes in comparison to parameterized kernels with a fixed, K filters . However, they are less desirable as the number of parameters grows with growing vertex set, and also these filters (parameters) are not localized.

The popular alternative is the polynomial filter given in Eqn: 2.11. The polynomial filters of the K-th order are exactly K-localized. However, extracting features with these polynomial filters are compute intensive owing to the dense-dense multiplication of the signal, x and the basis, U besides the running time complexity of the eigen decomposition $O(n^3)$.

$$\text{Non-parametric filter:} \quad g_\theta(\Lambda) \quad = diag(\theta) \tag{2.10}$$

$$\text{Polynomial filter} \quad g_\theta(\Lambda) \quad = \sum_{k=0}^{K} \theta_k \Lambda^k \tag{2.11}$$

In order to overcome this computationally expensive step, we can resort to approximate computations of polynomials such as that of Chebyshev polynomial approximations. A truncated series of Chebyshev polynomials, which can be defined recursively in terms of L (Eqn: 2.12) is used to approximate the Polynomial filter i.e we intend to approximate $g_\theta(L)$ directly over $g_\theta(\Lambda)$ as given in Eqn: 2.12 as seen in Eqn: 2.13. This avoids the need to compute the full set of eigenvectors and more importantly, we can now leverage sparse-dense multiplications to compute Lx as L is a sparse matrix. This reduced the complexity to the $O(K|E|)$

$$\begin{aligned} \mathbb{T}_0(x) &= x \\ \mathbb{T}_1(x) &= L^{-}x \\ \mathbb{T}_k(x) &= 2L^{-}\mathbb{T}_{k-1}(x) - \mathbb{T}_{k-2}(x) \end{aligned} \tag{2.12}$$

$$\sum_{k=0} \theta_k \Lambda^k \approx \sum_{k=0}^{K} \theta_k \mathbb{T}_k(L^-)x$$
$$\text{where } L^- = \frac{2}{\lambda_{max}} L - I_N \tag{2.13}$$

2.3.3 Chebyshev Neural Network

The Chebyshev neural network of (Defferrard *et al.* (2016)) is built by stacking multiple K^{th} order Chebyshev filters mentioned in Eqn: 2.13. Chebyshev Neural networks defined in Eqn: 2.14 was defined for extracting graph level features to make graph classification. Graph Coarsening layers are interleaved between Chebyshev layers to coarsen the graph iteratively, and graph pooling layer was defined to summarize information from all the nodes of the graph at every layer. All Chebyshev layers have ReLU as the activation function, i.e $\sigma_k = \text{ReLU}\ \forall k \in [1, K]$. The features from the final layer are sent through an additional graph classification layer.

$$\begin{aligned} h_0 &= X \\ h_{k+1} &= \sigma_k(g_\theta(L^K)h_k) \\ &= \sigma_k(\sum_{k=0}^{K} \theta_k \mathbb{T}_k(L^-)h_k) \end{aligned} \tag{2.14}$$

2.4 Graph Convolutional Networks

(Kipf and Welling (2016)) introduced Graph Convolutional Networks (GCNs) as stacked approximations of first-order Chebyshev filters. They are also multi-layer convolutional neural network where the convolutions are defined on a graph structure for the problem of Semi-Supervised node classification. The significant improvement over Chebyshev Neural Networks is that, in GCNs, filtering happens at every step of the propagation thereby allowing only relevant information to flow across neighborhoods. This also enables GCNs to learn complex non-linear feature basis.

The conventional two-layer GCNs which captures information up to the 2^{nd} hop neighborhood of a node can be reformulated to capture information up to any arbitrary

hop, K as given below in Eqn: 2.15.

$$\begin{aligned} h_0 &= X \\ h_k &= \sigma_k(\hat{L}h_{k-1}W_k), \ \ \forall\, k \in [1, K-1]. \\ Y &= \sigma_K(\hat{L}h_{K-1}W_L) \end{aligned} \tag{2.15}$$

GCNs were used for multi-class classification task with Rectified Linear Unit (ReLU) as the activation function, i.e $\sigma_k = \text{ReLU}\ \forall k \in [1, K-1]$ and a softmax label layer, $\sigma_K = softmax$. We can rewrite the GCN model in terms of $(K-1)^{\text{th}}$ hop node and neighbor features as below by factoring $\hat{L}$.

$$\begin{aligned} h_k &= \sigma_k((\hat{D}^{-\frac{1}{2}}I\hat{D}^{-\frac{1}{2}} + \hat{D}^{-\frac{1}{2}}A\hat{D}^{-\frac{1}{2}})h_{k-1}W_k) \\ &= \sigma_k(SUM(\hat{D}^{-1}h_{k-1},\ \hat{D}^{-\frac{1}{2}}A\hat{D}^{-\frac{1}{2}}h_{k-1})W_k) \end{aligned} \tag{2.16}$$

2.4.1 Relation to Weisfeiler-Lehman (WL) algorithm

GCNs can be viewed as continuous approximations of the node features obtained with 1-dim Weisfeiler-Lehman algorithm (Kipf and Welling (2016); Hamilton *et al.* (2017)). The Weisfeiler-Lehman (WL) algorithm defines a vertex color (label) refinement scheme that is used to obtain graph signatures to perform graph-isomorphism tests. The WL algorithm starts with an initial coloring (discrete features), for all nodes and uses a hashing function that aggregates neighbors' colors and produces a new color. This step is repeated multiple times until we get a stable coloring. Eqn: 2.17 computes the color of a node, i in k^{th} iteration based on a hashing of neighbors' colors from the $k-1^{\text{th}}$ step.

$$color_k[i] = hash(\sum_{(i,j)\in E} color_{k-1}[j]) \tag{2.17}$$

The WL-algorithm, at every step, k applies a local function to compute k-hop signatures/representations of nodes. At every step, the information from a node gets propagated further away as shown in Figure: 2.1. Figure: 2.1 shows information at the nodes at different step, with leftmost being the initial 0^{th} step and the rightmost being 2^{nd} step. At every step, each node aggregates neighbors symbols and takes a union of it.

GCN models are parameterized differentiable extensions of WL algorithm based

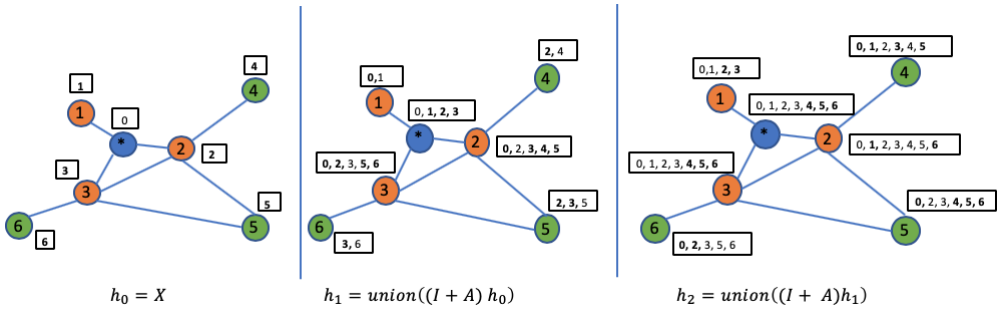


Figure 2.1: Information Propagation in 1-d Weisfieler Lehman algorithm

graph kernels, where the hashing function is replaced with a non-linear function, and the simple union aggregation function is replaced with differentiable functions such as mean with GCNs.

2.5 GraphSAGE

Graph SAmple and AggreGatE model (GraphSAGE) proposed in (Hamilton *et al.* (2017)) consists of 3 models with different differentiable neighborhood aggregator functions. GraphSAGE models were defined for the multi-label Semi-Supervised inductive learning task, *i.e* generalizing to unseen nodes during training. Let the function $Aggregate()$ abstractly denote the different aggregator functions in GraphSAGE, specifically Aggregate $\in$ {mean, max pooling, LSTM} and we will refer to these models as GS-MEAN, GS-MAX and GS-LSTM, respectively. Similar to GCN, GraphSAGE models also recursively combine neighborhood information at each layer of the Neural Network. GraphSAGE has an additional label layer unlike GCN, i. e., $h_L = h_{K+1}$. Hence, $\sigma_k = \text{ReLU}\ \forall k \in [1, K]$ and $\sigma_L = \text{sigmoid}$. Here, the weights $W_{\hat{k}} \in \mathbb{R}^{2d \times d}$.

$$
\begin{aligned}
h_0 &= X \\
h_k &= \sigma_k(\textsc{concat}(h_{k-1}, Aggregate(h_{k-1}, A)W_{\hat{k}}) \\
&\quad \forall k \in [1, K] \\
Y &= \sigma_{K+1}(h_K W_L)
\end{aligned}
\tag{2.18}
$$

GraphSAGE models, unlike GCN, are defined to work with partial neighborhood information. For each node, these models randomly sample and use only a subset of neighbors from different hops. This choice to work with partial neighborhood informa-

tion allows them to scale to large graphs but restricts them from capturing the complete neighborhood information. Rather than viewing it as a choice it can also be seen as restriction imposed by the use of Max Pool and LSTM aggregator functions which require fixed input lengths to compute efficiently. Hence, GraphSAGE constraints the neighborhood subgraph of a node to contain a fixed number of neighbors at each hop.

2.6 Iterative Collective classification Algorithm

Iterative Collective classification Algorithm (ICA) (Sen P *et al.* (2008)), is more of a framework that can incorporate relational information into any existing classifiers. ICA (Neville and Jensen (2000)) was the popular Collective Classification model in the last decade because of its generality and also primarily because it can handle non-homophilous relational information too. ICA explicitly tries to capture the correlation of a node's features with its neighbor's labels and hence does not make any assumptions on the label smoothness among the neighborhood. There are three major components of ICA are (i) neighborhood summaries, (ii) relational classifier and (iii) iterative inference. The three components are explained below:

- Neighborhood summaries: ICA leverages relational information of a node in terms of neighborhood summaries. Conventionally, some label statistics over the neighbors' labels are used. Few popular ones are the label sum, avg, min and max of the label counts. Since the majority of nodes are unlabeled in the SSL setup, a pre-requisite for ICA is to bootstrap the labels of the unlabeled nodes. A common and effective strategy is to bootstrap the labels with predictions from a non-relational classifier based on the attributes of the nodes.
- Relational classifier: The relational classifier of ICA can be any standard classifiers. ICA concatenates the obtained neighborhood summaries of nodes with their nodes' features to obtain a new relational node features. Then, this new node feature is used to learn a classifier. This classifier can now make predictions leveraging the relational features.
- Iterative inference: Since the labels of the unlabeled nodes are bootstrapped and are not the true labels. The neighborhood summaries based on these obtained estimates are not completely reliable. Hence, it is important to improve these estimates in order to make the label predictions better. The iterative inference component of ICA helps in achieving this by iteratively updating the label estimates. After learning a relational classifier with the bootstrapped labels for unlabeled nodes, ICA iteratively makes label estimates with the relational classifier based on the current label summaries of nodes. At every iteration, the label summaries are updated based on the current label estimates. In short, the ICA framework makes label predictions based on label summaries with the learned

relational classifier and then feeds back a new label summary computed with the recent label predictions made. The iterative refinement of label estimates are done till some fixed number of steps or till some convergence criteria is satisfied.

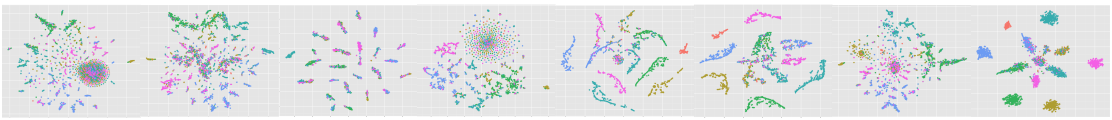
Better label estimates are obtained with ICA not only because of better neighborhood summaries but also because of label information propagation. With every iteration, a node's label estimate is propagated and used to make label update for another similar to 1-dim WL algorithm discussed in Fig: 2.1. Here, the relational classifier by itself is a local first-order classifier which can make label predictions based on the node and information from its immediate neighbors. The significant difference with ICA and recent neural network based models for graphs is that ICA cannot learn and end-end differentiable multi-hop representation for the end tasks. The learning happens before the information is propagated beyond one-hop. Moreover, another significant difference is the neural network models do not leverage label information. There is some variant of ICA which also exploits neighbor's attribute information but again not in an entirely end-end differentiable manner.

2.7 Semi-Supervised Non-Negative Matrix Factorization for clusterable graph embeddings (SS-NMF)

This work[1] focuses on learning Semi-Supervised node embeddings for non-attributed graphs under the Non-Negative Matrix Factorization (NMF) framework. The model proposed in this work, Semi-Supervised NMF (SS-NMF), is a novel node embedding model that learns Semi-Supervised cluster invariant representations. The learned node embeddings are densely clustered into regions of same or similar labels as can be seen with the t-Distributed Stochastic Neighbor Embedding (t-SNE) plots presented in Figure: 2.2. These clusterable node representations provide not only superior visualizations but also improved node classification results.

[1]This work was done in collaboration with Anasua Mitra, Ph.D. scholar at IIT Gauhati when she interned in R.I.S.E lab, IIT Madras. The contributions of the work will go into her thesis.

Figure 2.2: t-SNE Visualization of Embeddings on Citeseer Dataset for Unsupervised & Semi-Supervised Methods



(a) MFDW (b) MNMF (c) GEMSEC (d) COME (e) MFDW+Y (f) MNMF+Y (g) MMDW (h) SS-NMF

2.7.1 Notations for SS-NMF

Hence, we provide additional notations and also revise existing notations specific to this work.

Let the true label matrix, $Y \in 0, 1^{|S|*l}$ be expanded with zero rows to become $Y \in \mathbb{R}^{n*q}$. Let, $D(A)_{n\times n}$ be the diagonal degree matrix of the adjacency matrix A defined as $d_{\text{ii}} = \sum_j a_{\text{i,j}}$. Thus, the unnormalized Laplacian operator on the graph G can be defined as $\Delta(A)_{N\times N} = D(A) - A$. Let, $W \in \mathbb{R}^{n*n}$ refer to a penalty matrix that penalizes reconstruction of a matrix according to the value defined.

2.7.2 SS-NMF:Semi-Supervised NMF model

Here we incrementally build the proposed model step by step.

Encode local neighborhood invariance in node embeddings: The basic component of the model learns locally invariant node representations, i.e., nodes which are connected have similar representations. We obtain locally invariant representations by factorizing a proximity matrix, $P \in \mathbb{R}^{n*n}$ that encodes the similarity between the nodes. The model is similar to Matrix Factorized DeepWalk (MFDW) model with Pointwise Mutual Information matrix as the proximity matrix (Tu *et al.* (2016)) with the difference that herein the Non-negative Matrix Factorization (NMF) framework is used. NMF models are well suited as the proximity matrix is positive. Closely, following (Tu *et al.* (2016)), we only consider the first order and second order average transition probability between a pair of nodes to compute the P. Recent papers such as (Levy and Goldberg (2014)), (Yang *et al.* (2015)) and (Qiu *et al.* (2017)) suggest that DeepWalk is equivalent to factorizing Pointwise Mutual Information matrix that represents vertex-neighbor transition probability.

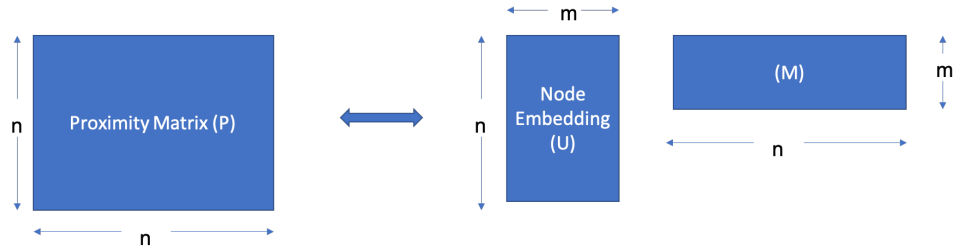


Figure 2.3: Unsupervised node embedding

We factorize the proximity matrix, P into two non-negative basis matrices - the node representation matrix $U \in \mathbb{R}^{m \times n}$ and the context neighbourhood representation matrix $M \in \mathbb{R}^{m \times n}$ as given below and depicted in Figure: 2.3,

$$\mathcal{O}_{network} = \min_{M,U} \|P - U^T M\|^2 : M \geq 0, U \geq 0 \tag{2.19}$$

Encode supervision knowledge: In order to learn the Semi-Supervised representations, we need to jointly factorize the label matrix, Y along with S as shown in Figure: 2.4. We define the label matrix factorization term in Eqn: 2.20. Where $W \in \mathbb{R}^{q \times n}$ is a weight penalty matrix that zeros out all the label information of test instances. Specifically, W_i is equal to 0 if the corresponding Y_i is unknown and 1 otherwise. $\odot$ is Hadamard or element-wise multiplication. $Q \in \mathbb{R}^{q \times m}$ is the label basis matrix. The supervision component is defined as follows:

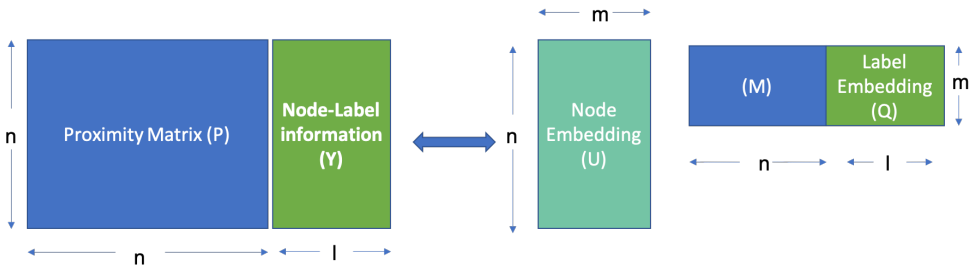


Figure 2.4: Semi-Supervised node embedding

$$\mathcal{O}_{label} = \min_{Q,U} \|W \odot (Y - QU)\|^2 : Q \geq 0, U \geq 0 \tag{2.20}$$

Encode local neighborhood invariance in label space: The model also includes the classic label smoothing Laplacian regularizer, $LS(S, QU)$. Label smoothing regularizer constraints the connected nodes to have similar labels, where QU is the predicted

(reconstructed) label matrix as defined in the previous component.

$$\mathcal{O}_{LL} = LS(S, QU) = Tr\{(QU)\Delta(S)(QU)^T\} \tag{2.21}$$

Encode Semi-Supervised cluster invariance: Here, we define our proposed novel component that allows for learning cluster invariant node representations. Unlike models which enforce explicit Laplacian regularization on the embedding space or label space, we enforce constraints on an abstract space, clusters. In essence, it allows the model to learn clusterable representations such that there is high label smoothness within the cluster. This component primarily comprises two components:

- **Learn cluster assignment via orthogonality constraint:** Let, $H \in \mathbb{R}^{k \times n}$ represents the binary cluster membership indicator matrix defined for k number of clusters. We obtain H by projecting node embeddings, U on cluster basis C, as $H = CU$. We can restrict the clusters to have different assignments and have less overlap with each other via block diagonal constraints. Here we resort to blocks of size one by enforcing the orthogonal constraints, which encourages every cluster to be different from each other. This soft-clustering criteria is enforced by $Trace(HH^T) = N$. We relax this constraint by setting $HH^T = I$, similar to (Wang *et al.* (2017)). The regularization weight for this is kept as a constant value 10^9.

- **Encode global context with local cluster invariance:** We enforce this constraint by applying Laplacian regularization on H with label similarity based proximity matrix, P_Y. We define the label similarity network defined over train data as $P_Y = (W \odot Y)^T(W \odot Y) \in \mathbb{R}^{n \times n}$, where $\Delta(P_Y) = D(P_Y) - P_Y$ is the unnormalized Laplacian operator on P_Y. With $\Delta(P_Y)$, we define the cluster smoothing Laplacian regularizer, $LS(P_Y, H)$.

  $$LS(P_Y, H) = Tr((H)\Delta(P_Y)(H)^T) \tag{2.22}$$

  The label based similarity matrix introduces new edges between nodes of similar labels which may be far away or not even connected in the original network, P. This allows clusters to enforce a global context.

Cluster invariant representations that enforce similar cluster assignments to nodes of same labels are obtained with the objective in Eqn: 2.23. There is a circular enforcement between H and U, i.e., U learns from H, H implicitly learns from U and H explicitly learns from the cluster regularization term & non-overlapping/ orthogonality constraint. Therefore, H pushes two nodes with the same labels and similar neighborhood structure

together into the same cluster and ensures that these two have similar representations.

$$L_{group} = \min_{H,C,U \geq 0} \beta \|H - CU\|^2 + \phi LS(P_Y, H) + \zeta \|HH^T - I\|^2 \tag{2.23}$$

SS-NMF Model In SS-NMF, the node representations, U are learned by jointly factorizing the local neighborhood proximity matrix S, label matrix Y, inferred cluster assignment matrix H and are indirectly influenced by smoothing on the label and cluster space. The joint objective is given below in Figure: 2.5 and in Eqn: 2.24.

$$\mathcal{O} = \alpha(\mathcal{O}_{network}) + \theta(\mathcal{O}_{label}) + \delta(\mathcal{O}_{LL}) + \mathcal{O}_{group} + \lambda(L2_{reg}) \tag{2.24}$$

$\alpha, \beta, \theta, \zeta, \phi, \delta, \lambda$ are hyper-parameters controlling the importance of respective terms in the equation. Since the joint non-negative constrained objective is not convex, we can iteratively solve the convex sub-problem for each of the factors $M, U, C, Q,$ & H as with most multiplicative NMF approaches.

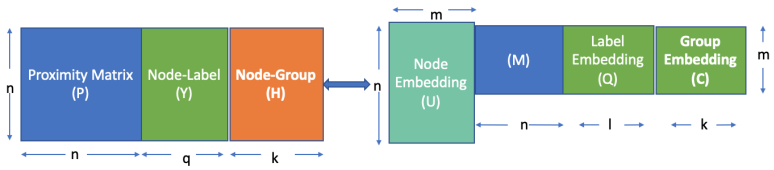


Figure 2.5: SS-NMF: Semi-Supervised NMF

2.7.3 Optimization

Let $\psi_1, \psi_2, \psi_3, \psi_4, \psi_5$ be the Lagrange multipliers for the non-negative constraints on factor matrices M, U, C, Q, H respectively. SS-NMF's loss function in Eqn: 2.24 can be expanded and rewritten with Lagrange multipliers for non-negative constraints as

follows:

$$\begin{aligned}\mathcal{L} &= \alpha Tr[PP^T - 2PM^TU + U^TMM^TU] \\ &+ \beta Tr[HH^T - 2HU^TC^T + CUU^TC^T] \\ &+ \theta Tr[W \odot \{YY^T - 2YU^TQ^T + QUU^TQ^T\}] \\ &+ \zeta Tr[HH^THH^T - 2HH^T + I] + \phi Tr\{H\mathcal{L}(P_Y)H^T\} \\ &+ \delta Tr\{(QU)\mathcal{L}(P)(QU)^T\} \\ &+ \lambda Tr(MM^T + QQ^T + CC^T + UU^T + HH^T) \\ &+ Tr[\psi_1 M^T + \psi_2 U^T + \psi_3 C^T + \psi_4 Q^T + \psi_5 H^T]\end{aligned}$$

The Karush−KuhnTucker (KKT) conditions for the non-negativity of the different terms are: $\psi_{1ab}m_{ab} = 0$, $\psi_{2ab}u_{ab} = 0$, $\psi_{3ca}c_{ca} = 0$, $\psi_{4da}q_{da} = 0$ and $\psi_{5cb}h_{cb} = 0$, where $M = [m_{ab}], U = [u_{ab}], C = [c_{ca}], Q = [q_{da}], H = [h_{cb}]$ s.t. a, b, c, d are the respective row & column indices. Following the KKT conditions, we can obtain the partial derivatives for each of the factors.

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial M} &= -2\alpha US + 2\alpha UU^TM + 2\lambda M + \psi_1 \\ \frac{\partial\mathcal{L}}{\partial C} &= -2\beta HU^T + 2\beta CUU^T + 2\lambda C + \psi_3 \\ \frac{\partial\mathcal{L}}{\partial Q} &= -2\theta(W \odot Y)U^T - 2\delta(QUS)U^T + 2\theta(W \odot QU)U^T \\ &+ 2\delta(QUD(S))U^T + 2\lambda Q + \psi_4 \\ \frac{\partial\mathcal{L}}{\partial U} &= -2\alpha MS^T - 2\beta C^TH - 2\theta Q^T(W \odot Y) - 2\delta(Q^TQU)S + 2\alpha MM^TU \\ &+ 2\beta C^TCU + 2\theta Q^T(W \odot QU) + 2\delta(Q^TQU)D(S) + 2\lambda U + \psi_2 \\ \frac{\partial\mathcal{L}}{\partial H} &= 2\beta H - 2\beta CU + 4\zeta HH^TH - 4\zeta H + 2\lambda H + 2\phi HD(E) - 2\phi HE + \psi_5\end{aligned}$$

From the partial derivates, we can derive the following multiplicative update rule for the factors.

$$M = M \odot (\frac{\alpha UP}{\alpha UU^TM + \lambda M}) \tag{2.25}$$

$$C = C \odot (\frac{\beta HU^T}{\beta CUU^T + \lambda C}) \tag{2.26}$$

$$Q = Q \odot (\frac{\theta(W \odot Y)U^T + \delta(QUP)U^T}{\theta(W \odot QU)U^T + \delta(QUD(P))U^T + \lambda Q}) \tag{2.27}$$

$$U = U \odot (\frac{\alpha MP^T + \beta C^T H + \theta Q^T(W \odot Y) + \delta(Q^T QU)P}{\alpha MM^T U + \beta C^T CU + \theta Q^T(W \odot QU) + \delta(Q^T QU)D(P) + \lambda U}) \tag{2.28}$$

$$H = H \odot (\frac{\beta CU + 2\zeta H + \phi HP_Y}{\beta H + 2\zeta HH^T H + \phi HD + \lambda H})^{1/4} \tag{2.29}$$

2.7.4 Experiments

In this section, we evaluate and analyze the proposed model, SS-NMF against the state-of-the-art models and numerous proposed adaptions of existing and new Semi-Supervised components for node classification. Additionally, we also demonstrate the superiority of the discriminative capacity of the learned embeddings by evaluating clusters against ground truth with Normalized Mutual Information (NMI) and t-SNE plots. Further, we also provide convergence plots and parameter sensitivity analysis.

Datasets

Description of the datasets used are provided below with summary statistics tabulated in Table: 2.1.

Table 2.1: Dataset statistics

V: Nodes, E: Edges, Y: Labels, ML: Multi-label dataset
D:Directed, W:Weighted, T:True, F:False

Dataset	$\|V\|$	$\|E\|$	$\|Y\|$	is ML?	D ǀ W	Type
Washington	230	596	5	F	F ǀ T	WWW
Wisconsin	265	724	5	F	F ǀ T	WWW
Texas	186	464	5	F	F ǀ T	WWW
Cornell	195	478	5	F	F ǀ T	WWW
Wiki	2,405	17,981	19	F	T ǀ F	Citation
Cora	2,708	5,278	7	F	T ǀ F	Citation
Citeseer	3,312	4,732	6	F	T ǀ F	Citation
Pubmed	19,717	44,338	3	F	F ǀ F	Citation
PPI	3,890	76,584	50	T	F ǀ F	Biological

WWW network: WebKB (Chakrabarti *et al.* (1998)) consists of four small web networks collected from four different universities - Washington, Wisconsin, Texas, and Cornell. These networks are a collection of web pages modeled as nodes with their hyperlinks forming the edges. The task here is to predict the type of webpage.

Citation networks: In the following citation networks, nodes are the research papers and edges denote a citation. (Cora Craven *et al.* (1998)), Citeseer (McCallum *et al.* (2000)), Wiki (Sen P *et al.* (2008)) and Pubmed are four bibliographic datasets where the task is to predict the research area of the papers.

Biological network: PPI (Stark *et al.* (2006)) is a Homo Sapiens' Protein-Protein Interaction (PPI) network where the objective is to predict the labels for nodes from the hallmark gene sets.

Baselines

The state of the art methods for Semi-Supervised classification on non-attributed graphs is limited only to MaxMargin-DeepWalk (MMDW) (Tu *et al.* (2016)) and Planetoid (Yang *et al.* (2016*a*)). Besides, we also introduce two more Semi-Supervised baselines built on top of the unsupervised community preserving embedding model (MNMF) (Wang *et al.* (2017)) and the Non-negative Matrix Factorization version of DeepWalk (*NMF:P*) (Tu *et al.* (2016)). They are *MNMF+Y* and *NMF:P+Y* respectively in which the original objectives are jointly factorized with label matrix (Y) as in Eqn: 2.20. The nomenclature for NMF based models will indicate that they are NMF along with the matrices being factored. For example, *NMF:P+Y* indicates that the matrix factorizes proximity matrix, P and label matrix Y.

For the sake of reference, we have also included the results of the original random walk sampling based DeepWalk (*DW*) (Perozzi *et al.* (2014)). Apart from *MNMF* (Wang *et al.* (2017)), we have also included few recent unsupervised community/cluster enhanced node representation learning models, viz, Community Embedding *ComE* (Cavallari *et al.* (2017)) and Graph Embedding with Self Clustering *GEMSEC* (Rozemberczki *et al.* (2018)) as baselines. *Planetoid* was defined for multi-class classification problem with stratified labeled train-set (which can be unrealistic). We empirically observed Planetoid-G (Planetoid for non-attributed graphs) perform poorly in comparison to other baselines where the labeled set is randomly drawn, and it is not directly ex-

tensible for the multi-label dataset (PPI). Hence, we do not report results for Planetoid here.

These above-discussed models will be the standard baselines that will be compared against SSNMF. Additional models used for experiments will be described in their respective sections.

Experiment Setup

The Semi-Supervised Learning experiment is set up with 50% labeled data, and the data is drawn randomly to create a 5 fold cross validation setup. All classification and clustering results reported here are an average over these five sets. For all the competing algorithms we set dimensions of the node embeddings as 128. We also extensively searched for optimal hyperparameter values for all the competing methods using 20% of the training data as a validation set.

Metrics: We report classification performance with Micro-F1 scores. Additionally, we define two aggregate metrics to measure the overall performance of models across datasets viz: *Rank* and *Shortfall*. *Rank* of a model is defined as the average position of the model when the results are ordered in descending order in each dataset and *Shortfall* of the model is defined as the average normalized difference from the best performing model in each dataset, see section: 3.9.1 for more details. The lower the rank and shortfall, the better is the performance of the model. The proposed models are the winner in all the experiments. However, we resort to these two aggregate scores to measure the consistency of models across datasets. We also provide statistical significance with Wilcoxon signed rank test, the established test for comparing two models on multiple datasets.

classifier: For all models, we learn an external Logistic Regression classifier to make label predictions from the models' learned node representations. Though we can obtain label predictions internally for the supervised models by reconstructing the label matrix, we found that using an external classifier further improves the performance in all models.

Table 2.2: The wide and narrow ranges of hyperparameters. A: NMF:P, B: NMF:P+Y, C: MMDW, D: MNMF, E: MF-Planetoid, F: SS-NMF

Factorization based methods	A	B	C	D	E	F	G	Hyperparameter range
α	True	True	True	True	True	True	True	$[0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.3, 0.5, 0.7, 1.0, 3.0, 5.0, 7.0, 10.0]$
β				True	True		True	
θ		True			True	True	True	
γ				True	True			
ϕ						True	True	
ζ				True	True	True	True	$1e+9$, $[1e+(5-10)]$
η			True					$[-2, -3, -4, -5]$
λ	True	True	True	True	True	True	True	$[0.001, 0.01, 0.1, 1.0, 10.0]$
k				True	True		True	25 values for each dataset by varying k with an increment of 2 in the upper range and with a decrement of 1 in the lower range from its actual number of labels q (inclusive)
Random-walk based methods	DeepWalk & Node2Vec		p & q = $[0.25, 0.50, 0.75, 1.0, 1.25, 1.50, 1.75, 2.0, 2.25, 2.50, 2.75, 3.0, 3.25, 3.50, 3.75, 4.0]$					walk-length = $[80]$ num-walks = $[40, 80]$
maximum iterations = 500, latent dimension = 128, initialization = âĂIJrandomâĂİ, convergence = $1e-5$, 5 fold cross validation, 10% & 50% test-train split								

Table 2.3: Node Classification Results | Micro-F1 Scores

	Unsupervised Models					Semi-Supervised Models			
			Cluster/ Community Models			Existing	Proposed Baseline Variants		Proposed
Datasets	DW	NMF:P	MNMF	GEMSEC	ComE	MMDW	NMF:P+Y	MNMF+Y	SS-NMF
Cora	80.148	80.369	82.664	77.063	82.030	83.916	83.690	84.546	**86.273**
Citeseer	57.272	59.710	63.565	56.403	60.712	68.911	68.618	69.005	**70.141**
Wiki	63.009	63.940	65.752	58.969	63.840	66.687	66.177	66.750	**70.906**
Washington	59.130	59.130	62.609	60.913	62.087	61.130	62.609	62.962	**66.087**
Wisconsin	48.120	49.023	51.128	52.030	51.504	50.376	50.376	52.759	**56.391**
Texas	58.511	56.383	57.447	57.021	59.043	56.383	58.511	57.447	**63.830**
Cornell	38.776	50.000	51.448	51.277	49.490	51.224	50.940	52.041	**53.061**
PPI	22.216	21.753	21.235	21.417	22.373	**23.579**	22.186	21.452	23.433
Pubmed	77.413	79.077	79.837	80.599	81.626	80.776	82.388	83.001	**84.048**
Rank	7.333	7.222	5.222	6.889	5.000	4.444	4.222	2.778	**1.111**
Shortfall	0.84	0.729	0.553	0.743	0.513	0.41	0.406	0.381	**0.007**

Node classification

Node classification results are reported in Table: 2.3. The bolded entries in a dataset column denote the best score achieved in that dataset, and the underlined entries denote the second best score.

Unsupervised Models:

NMF based neighborhood embedding model, *NMF:P* performs similar or better than the sampling-based DW as shown in Tu *et al.* (2016) on all but Texas and PPI dataset. Both these models which do not encode supervision information or any clustering information are the two least performing models as depicted by their Ranks. The superiority of *NMF:P* over *DW* is visible from the relatively lower Shortfall score. This is consis-

tent with findings in the literature that typically Matrix factorization models are better. *NMF:P* is not significantly different from *DW* for $p < 0.05$ on two-tailed Wilcoxon signed-rank tests. Skip-gram based models additionally have the power of non-linearity and optimize cross entropy loss when compared to the simple linear squared-error based reconstruction loss used in our NMF models. We believe factorizing with KL-divergence over squared error might improve the performance of factorization models and match DW in datasets where they currently fail.

Cluster (cluster) enforcing models are the superior unsupervised models. Among them, *MNMF* outperforms *GEMSEC* and *ComE* on 6/9 datasets except for Texas, PPI, and Pubmed where *ComE* outperforms *MNMF* significantly. This is reflected in the aggregate scores. Despite *ComE* beating *MNMF* on a lower number of datasets, its rank is better than *MNMF* as on those datasets it beats both *MNMF* and other models significantly. It is ranked 2$^{\text{nd}}$ in Texas, 3$^{\text{rd}}$ in PPI and 4$^{\text{th}}$ in Pubmed. On the other hand, looking at the shortfall score, *MNMF* seems to be better than *ComE* slightly as on multiple datasets it beats *ComE* by $2-3$ points. *GEMSEC* seems to be poor among the community models. *MNMF* is not significantly different from *ComE* for $p < 0.05$ on two-tailed Wilcoxon signed-rank tests (Demšar (2006)), this reflects the shortfall scores.

Semi-Supervised Models:

All supervised models obtain better ranking and lower shortfall over unsupervised models. The Semi-Supervised (SS) variants of the unsupervised models are (statistically) better than their unsupervised counterparts on all datasets, i.e., *MNMF+Y* > *MNMF* and *MFDW+Y* > *MFDW*. Among the baselines, the proposed variant, MNMF+Y is the second best on all but PPI dataset where MMDW is best. Learning community invariant representations in a supervised manner seems to be useful. Among *MMDW* and *NMF:P+Y*, there is no statistical difference. This is reflected in the obtained rank and shortfall scores which are similar.

The proposed model, SS-NMF *outperforms its base model*, *NMF:P* on all datasets which is a clear indicator that learning cluster and label invariant representations are useful. *SS-NMF is ranked first in* $8/9$ *datasets* while being ranked second on PPI. Thus, obtaining an average rank score of 1.11 and the lowest shortfall of 0.007. On the datasets where *SS-NMF* is ranked first, as per the paired t-test, there exists no case where

$p-value < 0.05$ and t-scores are positive (i.e., no competing method significantly beats *SS-NMF*).

Ablation study of SS-NMF

Here, we drill down the components of *SS-NMF* to analyze the importance of utilizing label information and cluster information.

Importance of Label Information: Since the smoothing on the cluster space is based on the label similarity graph, it is necessary to verify whether we need to factorize the label matrix and whether we should smooth the label space additionally. We report results for these study in Table: 2.4 where we remove these two components. The model in column 2 does not have the label smoothing term and the model in column 3 does not have the label factorization and label smoothing terms. Note that we cannot smooth on label space without predicting labels.

Label smoothing term provides an improvement of up to 3.88 points in Wiki and no improvement in Washington. On four datasets it offers an improvement over 1%, and on another two data sets, it provides an improvement less than 0.5%. From this, it is clear that the label smoothing term is helpful. Moreover, it can be seen from column 3 that removing label factorization, Y has a significant impact on *SS-NMF*. Removing label factorization results in an average drop of 3.45% and the performance up to 6% with the wiki. Whereas, the average reduction in performance when we remove label smoothing is 1.44. This experiment indicates that it is necessary to have both the terms.

Table 2.4: Importance of Label Information

	SS-NMF	- LS(P, Y)	- (Y + LS(P, Y))
Cora	86.273	85.838	82.411
Citeseer	70.141	69.750	65.531
Wiki	70.906	67.019	64.689
Washington	66.087	66.087	65.217
Wisconsin	56.391	54.136	54.009
Texas	63.830	61.702	59.575
Cornell	53.061	52.041	51.022

Importance of Clustering Information: Here, we try to understand the importance of the proposed cluster smoothing term by removing the cluster smoothing term first and then additionally removing the cluster assignment term too. This corresponds to column

2 and column 3 in Table: 4 respectively. Removing the cluster smoothing term results in an average drop of 0.8% across datasets whereas completely removing all cluster related components results in an average drop of 1.17%. Cluster smoothing term provides an additional improvement of up to 1.5% on three datasets and > 0.5 improvement on $5/7$ datasets. Removing the cluster term results in a drop in performance by more than 2.2% on three datasets. Note that random cluster assignments without any clustering objective to drive the learning outperforms the second best model, *MNMF+Y* in Table 2.3. This demonstrates the usefulness of encoding local invariant representations on label space. In the next section, we further clearly see the benefit of combining label and cluster smoothing terms.

Table 2.5: Importance of Cluster Information

	SS-NMF	- LS(P_Y, H)	- (H + LS(P_Y, H))
Cora	86.273	86.119	86.052
Citeseer	70.141	69.584	69.523
Wiki	70.906	70.490	70.490
Washington	66.087	65.217	63.478
Wisconsin	56.391	54.887	54.135
Texas	63.830	62.766	62.766
Cornell	53.061	52.041	52.041

Semi-Supervised Learning Study

In this section, we analyze and compare different Laplacian smoothing in Table: 2.6. First, four models in columns involve Laplacian smoothing term based on the proximity graph, P and the next four models involve Laplacian smoothing term based on the label similarity graph, P_Y. All the models in this table additionally factorize label matrix, Y. The model in the last column is *SS-NMF*. *SS-NMF is the winner across the board on all datasets with Rank 1 and Shortfall 0.*

Model in column 2, *NMF: LS(P, Y) +Y*, is the standard Label Propagation (LP) implemented in NMF style (*NMF: LP*). It does not additionally factorize the proximity matrix, P and thus does not have any network embeddings. This is the worst performing model with the highest rank and shortfall. *From this model, it is evident that we need to learn network embedding as well.*

Model in column 1, *NMF: LS(P, U) +Y* learns embeddings such that the embeddings of neighboring nodes are closer. This model is similar in spirit to *NMF: S + L* which

factorizes the proximity matrix directly. This model has the second highest shortfall.

Model in column 5, *NMF: LS(P_Y, U) + S + Y* (NMF: Planetoid) has the same objective as Planetoid-G but in NMF style. It enforces nodes with similar labels to have similar embeddings. In a head-on comparison with *NMF: LS(P, U) +Y* (col:2), it shows that *enforcing smoothness from a global context has remarkable improvement* in shortfall scores and is statistically better with $p < 0.02$.

Model in column 4, *NMF:P + LS(P, Y) +Y*, is the second best performing model on 5/7 datasets. This model can be seen as an improved Label propagation model that additionally factorize a proximity matrix, P to learn node embeddings that capture network structure. This model beats the base *NMF:LP*.

Model in column 6, is an extension of *NMF: Planetoid* which includes label smoothing. *It can be seen that adding label smoothing provides improvement on all datasets up to* 2.7 *points.*

Model in column 7 is *SS-NMF* without, label smoothing term. Label smoothing term is crucial as it gives improvements up to 3.8 points. Moreover, similarly semi-supervised clustering with global context is also important as it provides an improvement of up to 2.6 points over *NMF: S + LS(P, Y) +Y* (col:4) in Washington. *Adding clustering components to label smoothening term* NMF:P + LS(P, Y) +Y *provides an average improvement of* 1.3 *point.*

Table 2.6: Semi-Supervised Learning Analysis | Micro-F1 Scores

Laplacian	Neighborhood Proximity Graph (P)				Label Similarity Graph (P_Y)			
Invariant Space	Embedding: U	Label: Y	Cluster: H	S + LS(P, Y)	Embedding: U		Cluster: H(U)	
					+ S	+(S + LS(P, Y))	+ S	+(S + LS(P, Y))
Cora	85.021	84.945	84.502	86.052	84.384	85.230	85.838	**86.273**
Citeseer	69.163	68.859	69.342	69.523	69.523	69.897	69.750	**70.141**
Wiki	70.085	67.332	67.332	70.490	66.422	69.127	67.019	**70.906**
Washington	57.391	53.913	64.348	63.478	62.831	63.478	**66.087**	**66.087**
Wisconsin	48.872	47.368	52.632	54.135	52.132	54.135	54.136	**56.391**
Texas	57.447	56.383	59.575	62.766	59.362	60.638	61.702	**63.830**
Cornell	52.041	50.000	52.041	52.041	52.041	**53.061**	52.041	**53.061**
Rank	5.571	7.286	4.857	2.857	5.857	3.143	3.143	**1**
Shortfall	0.621	0.929	0.546	0.223	0.594	0.29	0.324	**0.000**

Clusterability of learned representations

We validate superior clusterability of the learned node representations quantitatively in Table: 2.7 and qualitatively with t-SNE plots in the next section.

In Table: 2.7 we report the averaged cluster quality of learned embeddings. We take the embeddings learned for node classification and report averaged NMI results over five folds, where each of these fold were run five more times with different initializations. We used different initialization techniques for initializing the mean of clusters (k-means++, Principal Component Analysis (PCA)). The clusters were obtained with k-means and Fuzzy c-means algorithms for multi-class and multi-label datasets correspondingly. The optimal number of clusters was obtained using elbow criterion and gap statistics (Tibshirani *et al.* (2001)). We evaluate the obtained clusters against gold standard classes and report the NMI scores. We used Overlapping NMI (Lancichinetti *et al.* (2009)), (McDaid *et al.* (2011)) to evaluate overlapping clusters corresponding to the multi-label datasets.

Table 2.7: Node Clustering | (O)NMI Scores

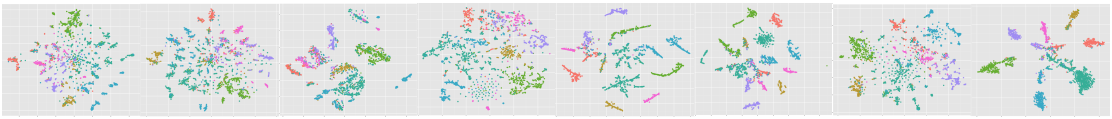
	Unsupervised Models					Semi-Supervised Models			
			Cluster/ Community Models			Existing	Proposed Baseline Variants		Proposed
	DW	NMF:P	GEMSEC	ComE	MNMF	MMDW	NMF:P + Y	MNMF+ Y	SS-NMF
Cora	34.28	34.40	35.83	41.02	39.29	50.31	51.38	53.21	**57.93**
Citeseer	19.04	17.71	21.42	24.42	29.96	32.70	28.94	41.19	**53.32**
Wiki	32.57	28.31	33.86	32.59	45.62	33.82	47.80	48.38	**61.06**
Washington	2.88	9.93	8.98	5.89	19.90	15.78	18.45	33.52	**40.86**
Wisconsin	5.04	6.09	5.46	5.22	11.20	9.27	6.81	17.89	**33.9**
Texas	2.70	2.85	2.35	3.65	9.00	7.99	10.61	15.14	**35.56**
Cornell	3.53	4.16	3.91	3.35	3.99	**8.76**	4.49	4.14	5.88
PPI	9.44	7.91	9.63	9.07	8.77	8.44	8.26	9.19	**11.48**
Pubmed	20.15	17.28	19.93	29.83	29.77	28.56	29.39	37.32	**38.47**
Rank	7.67	7.44	6.56	6.33	4.44	4.56	4.33	2.56	**1.11**
Shortfall	0.914	0.956	0.864	0.817	0.678	0.6	0.641	0.427	**0.059**

From Table: 2.7, it is evident that *SS-NMF* performs well on Semi-Supervised node clustering task. It is the best performing model on eight datasets where it beats the second best model by 1-7%, and it is the second best performing model on the other two datasets where it is falling short of the best by a mere 0.1%. All the Semi-Supervised NMF models outperform the unsupervised NMF models except for *MMDW* which is outperformed by *MNMF*. Both *MFDW+Y* and *MNMF+Y* outperform their unsupervised counterparts, *MFDW* and *MNMF*. The supervised *MMDW* outperforms the simple unsupervised *MFDW* in all but Cornell. However, it is thoroughly washed out in comparison against the unsupervised *MNMF*. Though *MMDW's* max-margin representations outperformed unsupervised *MNMF*, it seems that they are not well clusterable. The consistent superior performance of *SS-NMF* & *MF-Plan* suggests that the label similarity based clusterability criteria can learn informative node representations beyond the graph structure. This is supported by the t-SNE plots too, especially that

of *SS-NMF* which provides superior high-quality visualizations of well separable homophilous clusters.

tSNE Visualization

Figure 2.6: t-SNE Visualization of Embeddings on Cora Dataset for Unsupervised & Semi-Supervised Methods



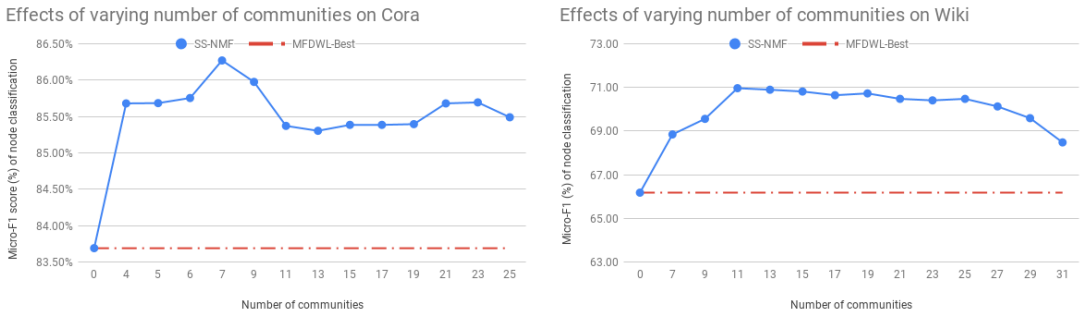
(a) MFDW (b) MNMF (c) GEMSEC (d) COME (e) MFDW+Y (f) MNMF+Y (g) MMDW (h) SS-NMF

Here, we present the details of t-SNE experiment on the learned node embeddings for Citeseer and Cora dataset in Figure: 2.2 & 2.6. t-SNE plots are especially well-suited for the visualization of high dimensional data. As t-SNE algorithm scales quadratically in terms of the number of nodes N, we first reduced the dimension of learned node embeddings to 64 retaining as much information as possible using the PCA algorithm. Next, we feed this compressed data to the t-SNE algorithm. In t-SNE, the perplexity term controls the number of neighbors for each sample to take into consideration while preserving the local structure in the reduced dimension space. We experimented with perplexity in the range of $10 - 100$, increasing by a step size of 10. We found that perplexity did not have a significant effect on the visualizations for values > 30. Hence, we fixed 40 as a common value of perplexity for all the competing methods. It can be seen that our proposed model obtains better clusters visually compared to other SS methods.

Varying Number of Clusters

We performed a study to understand how the number of clusters influenced the node classification performance and whether there is any need for learning clusters at all. For Cora and Wiki, we varied the number of clusters as in Figure: 2.7. Blue solid lines indicate the node classification performance of our proposed method. Corresponding Red dotted horizontal lines represent respective performances where all the cluster related terms were set to 0 ($\beta = 0, \phi = 0, \zeta = 0$), i.e.- learning no clusters (*NMF:P*

Figure 2.7: Varying Number of Clusters



scores). As we can see, major portions of the curves are above their respective dotted lines, indicating that learning clusters help in guiding node representations.

CHAPTER 3

Higher Order Propagation Framework

The graph propagation based neural networks are based on differentiable extensions of the popular Weisfieler-Lehman(WL) kernels. In this chapter, we first show that a direct adaptation of WL kernels for Collective Classification is inherently limited as node features get exponentially morphed with neighborhood information when considering farther hops. More importantly, learning to aggregate information from K-hop neighborhood in an end-to-end differentiable manner is not easily scalable. The exponential increase in neighborhood size with increase in hops severely limits the model due to excessive memory and computation requirements. Herein, we propose a Higher-order Propagation framework (HOPF) that provides a solution for both these problems.

The remainder of this chapter is as follows: We first provide a generic graph kernel that encompasses numerous existing and our proposed models for collective Classification as different instantiations of it. Then, we discuss a hitherto undiscovered phenomena in existing differentiable graph kernels termed as the Node Information Morphing issue. Then, we propose a specific instantiation of the generic kernel called the Node Information preserving kernel that solves this issue. Followed by this, we propose the Higher Order Propagation Framework, HOPF as framework that combines the generic graph kernel with an iterative inference mechanism. Such a combination of classical iterative inference mechanism with recent differentiable kernels allows the framework to learn graph convolutional filters that simultaneously exploit the attribute and label information available in the neighborhood. The iterative learning and inference component scales the differentiable graph kernels to larger hops beyond the memory limitations. Finally, we provide the details of the experiments followed by results and analysis of the issues of interest and proposed solutions on datasets across diverse domains.

Models	Φ_k	F(A)	Ψ_k	α	β	$W_k^\phi = W_k^\psi$?	Differentiable Kernel	Iterative Inference
BL_NODE	h_0	-	-	1	-	-	-	No
BL_NEIGH	-	$D^{-1}A$	h_{k-1}	-	1	-	Yes	No
SS-ICA	h_0	$D^{-1}A$	$\hat{Y}$	1	1	No	No	Yes
WL	h_{k-1}	A	h_{k-1}	1	1	-	-	No
GCN	h_{k-1}	$(D+I)^{-1/2}A(D+I)^{-1/2}$	h_{k-1}	$(D+I)^{-1}$	1	Yes	Yes	No
GCN-MEAN	h_{k-1}	$D^{-1}A$	h_{k-1}	1	1	Yes	Yes	No
GS-Pool	h_{k-1}	$maxpool$	h_{k-1}	1	1	No	Yes	No
GS-MEAN	h_{k-1}	$D^{-1}A$	h_{k-1}	1	1	No	Yes	No
GS-LSTM	h_{k-1}	LSTM gates	LSTM	1	1	No	Yes	No
NIP-MEAN	h_0	$D^{-1}A$	h_{k-1}	1	1	No	Yes	No
I-NIP-MEAN	h_0	$D^{-1}A$	$h_{k-1}, \hat{Y}$	1	1	No	Yes	Yes

Table 3.1: Baselines, existing and proposed models seen as instantiations of the proposed framework.

3.1 Generic propagation kernel

We define the generic propagation (graph) kernel as follows:

$$\begin{aligned} h_0 &= X \\ h_k &= \sigma_k(\alpha \cdot (\Phi_k \cdot W_k^\phi) \ + \ \beta \cdot (F(A) \cdot \Psi_k \cdot W_k^\psi)) \end{aligned} \tag{3.1}$$

where Φ_k and Ψ_k are the node and neighbor features considered at the k^{th} propagation step (layer), $F(A)$ is a function of the adjacency matrix of the graph, and W_k^ϕ and W_k^ψ are weights associated with the k-th layer of the neural network. One can view the first term in the equation as processing the information of a given node and the second term as processing the neighbors' information. The kernel recursively computes the outputs of the k^{th} layer by combining the *features* computed till the $(k-1)^{\text{th}}$ layer. σ_k is the activation function of the k-th layer and α and β can be scalars, vectors or matrices depending on the kernel.

Label predictions, $\hat{Y}$ can be obtained by projecting h_K onto the label space followed by a sigmoid or softmax layer corresponding to multi-class or multi-label classification task. The weights of the model are learned via backpropagation by minimizing an appropriate classification loss on $\hat{Y}$.

3.1.1 Relation to existing works:

Appropriate choice of α, β, Φ, Ψ and $F(A)$ in the generic kernel yield different models. Table 3.1 lists out the choices for some of the popular models, as well as our proposed approaches. Iterative collective inference techniques, such as the ICA family combine node information with aggregated label summaries of immediate neighbors to make predictions. Aggregation can be based on averaging kernel: $F(A)$=$D^{-1}A$, or label count kernel: $F(A)$=A, etc with labels as neighbors features (Ψ_k=$\hat{Y}$). This neighborhood information is then propagated iteratively to capture higher order information. ICA also has a SSL variant (McDowell and Aha, 2012) where after each iteration the model is re-learned with updated labels of neighbors. Table: 3.1 shows how the modular components can be chosen to see Semi-Supervised ICA (SS-ICA) as a special instantiation of our framework.

The Weisfeiler-Lehman (WL) family of recursive kernels (Weisfeiler and Lehman, 1968; Shervashidze *et al.*, 2011) were initially defined for graph isomorphism tests and most recent CC methods use differentiable extensions of it. In its basic form, it is the simplest instantiation of our generic propagation kernel with no learnable parameters as shown in Table: 3.1.

The normalized symmetric Laplacian kernel (GCN) used in Kipf and Welling (2016) can be seen as an instance of the the generic kernel with node weight, α=$(D+I)^{-1}$, individual neighbors' weights' $F(A)$=$(D+I)^{-1/2}A(D+I)^{-1/2}$, $\Phi_k = \Psi_k$ and $W_k^{\phi} = W_k^{\psi}$. We also consider its mean aggregation variant (GCN-MEAN), where $F(A) = D^{-1}A$. In theory, by stacking multiple graph convolutional layers, any higher order information can be captured in a differentiable way in $O(K \times E)$ computations. However in practice, the proposed model in Kipf and Welling (2016) is only full batch trainable and thus cannot scale to large graph when memory is limited.

GraphSAGE (GS) (Hamilton *et al.*, 2017) is the recent state-of-the-art for inductive learning. GraphSAGE has also proposed variants of k^{th} order differentiable WL kernels, viz: GS-MEAN, GS-Pool and GS-LSTM. These variants can be viewed as special instances of our generic framework as mentioned in the Table 3.1. GS-Pool applies a max-pooling function to aggregate neighborhood information whereas GS-LSTM uses a LSTM to combine neighbors' information sequenced in random order similar to the

model in Moore and Neville (2017). GS has a mean averaging variant, similar to the to GCN-MEAN model, but treats nodes separately from its neighbors, i.e $W_k^{\phi} \neq W_k^{\psi}$. Finally, it either concatenates or adds up the node and neighborhood information. GS-LSTM is over-parameterized for small datasets. With GS-MAX and GS-LSTM there is a loss of information as Max pooling considers only the largest input and LSTM focuses more on the recent neighbors in the random sequence.

3.2 Node Information Morphing (NIM): Analysis

In this section, we show that existing models which extract relational features, h_k do not retain the original node information, h_0 completely. With multiple propagation steps the h_0 is decayed and morphed with neighborhood information. We term this issue as Node Information Morphing (NIM).

For ease of illustration, we demonstrate the NIM issue by ignoring the non-linearity and weights. Based on the commonly observed instantiations of our generic propagation kernel (Eqn: 3.1), where $\Phi_k = \Psi_k = h_{k-1}$, we consider the following equation:

$$h_k = \alpha * I h_{k-1} + \beta * F(A) h_{k-1} \tag{3.2}$$

On unrolling the above expression, one can derive the following binomial form:

$$\begin{aligned} h_k &= (\alpha * I + \beta * F(A)) h_{k-1} \\ h_k &= (\alpha * I + \beta * F(A))^k h_0 \end{aligned} \tag{3.3}$$

From Eqn: 3.3, it can be seen that the relative importance of information associated with node's 0^{th} hop information, h_0, is $\frac{\alpha^k}{(\alpha+\beta)^k}$. Hence, for any positive β the importance of h_0 decays exponentially with k. It can be seen that the decay rate for GCN is $(D + I)^{-k}$ and $(2)^{-k}$ for the other WL kernel variants mentioned in Table: 3.1.

Skip connections and Node Information Morphing:
It can be similarly derived and seen that the information morphing not only happens at h_0 but also for every $h_k \forall k \in [0, K-1]$. This decay of neighborhood information can be lessened by leveraging skip connections. Consider the propagation kernel in Eqn:

3.2 with skip connections as shown below:

$$h_k = (\alpha * Ih_{k-1} + \beta * F(A)h_{k-1}) + h_{k-1} \tag{3.4}$$

The above equation on expanding as above gives:

$$h_k = ((\alpha + 1) * I + \beta * F(A))^k h_0 \tag{3.5}$$

The relative importance of weights of h_0 then becomes $\frac{(\alpha+1)^k}{(\alpha+\beta+1)^k}$, which decays slower than $\frac{\alpha^k}{(\alpha+\beta)^k}$ for all $\alpha, \beta > 0$. Though this helps in retaining information longer, it doesn't solve the problem completely. Skip connections were used in GCN to reduce the drop in performance of their model with multiple hops. The addition of skip connection in GCN was originally motivated from the conventional perspective to avoid reduction in performance with increasing neural network layers and not with the intention to address information morphing. In fact, their standard 2 layer model cannot accommodate skip connections because of varying output dimensions of layers. Similarly, GraphSAGE models which utilized concatenation operation to combine node and neighborhood information also lessened the decay effect in comparison to summation based combination models. This can be attributed to the fact that concatenation of information from the previous layer can be perceived as skip connections, as noted by its authors. Though the above analysis is done on a linear propagation model, this insight is applicable to the non-linear models as well. Our empirical results also confirm this.

3.3 Node Information Preserving models

To address the NIM issue, we propose a specific class of instantiations of the generic kernel which we call the Node Information Preserving (NIP) models. One way to avoid NIM issue is to explicitly retain the h_0 information at every propagation step as in the equation below. This is obtained from Eqn: 3.1 by setting $\Phi_k = h_0$ and $\Psi_k = h_{k-1}, \forall k$.

$$h_k = \alpha h_0 W_k^{\phi} + \beta F(A) h_{k-1} W_k^{\psi} \tag{3.6}$$

For different choices of α, β and $F(A)$, we get different kernels of this family. In

particular, setting $\beta = 1 - \alpha$ and $F(A) = D^{-1}A$ yields a kernel similar to Random Walk with Restart (RWR) (Tong *et al.*, 2006).

$$h_k = \alpha h_0 + \beta F(A) h_{k-1} \tag{3.7}$$

The NIP formulation has two significant advantages: (a) It enables capturing correlation between k-hop reachable neighbors and the node explicitly and (b) it creates a direct gradient path to the node information from every layer, thus allowing for better training. We propose a specific instantiation of the generic NIP kernel below:

$$\text{NIP-MEAN} : h_k = \sigma(\boldsymbol{h_0} W_k^{\phi} + D^{-1} A h_{k-1} W_k^{\psi}) \tag{3.8}$$

NIP-MEAN is similar to GCN-MEAN but with $\Phi_k = h_0$ and $W_k^{\phi} \neq W_k^{\psi}$.

3.4 Higher Order Propagation Framework: HOPF

Building any end-to-end differentiable model requires all the relational information to be in memory. This hinders models with a large number of parameters and those that process data in large batches from effectively utilizing the Graphics Processing Unit (GPU). For graphs with high link density and a power law degree distribution, processing even 2^{nd} or 3^{rd} hop information becomes infeasible. Even with p-regular graphs, the memory grows at $O(p^K)$ with the number of hops, K. Thus, using a differentiable kernel for even a small number of hops over a moderate size graph becomes infeasible.

To address this critical issue of scalability, we propose a novel Higher Order Propagation Framework (HOPF) which incorporates an iterative mechanism over the differentiable kernels. In each iteration of HOPF, the differentiable kernel computes a C hop neighborhood summary, where $C < K$. Every iteration starts with a summary, (Ξ^{t-1}), of the information computed until the $(t-1)$ step as given below.

$$\begin{aligned} h_0^0 &= X; \quad \Xi^0 = 0 \\ h_k^t &= \sigma(\alpha * \Phi_k W_k^{\phi} + \beta * F(A) \Psi_k^t W_k^{\psi}) \\ \Psi_k^t &= [\Psi_k, \Xi^{t-1}] \end{aligned} \tag{3.9}$$

After T iterations the model would have incorporated $(K = T \times C)$ hop neighborhood information. Here, we fix T based on the required number of hops we want to capture, K, but it can also be based on some convergence criteria on the inferred labels. For the empirical results reported in this work, we have chosen Ξ^{t-1} to be (predicted) labels $\hat{Y}$, along the lines of the ICA family of algorithms. Other choices for Ξ^{t-1} includes the K hop relational information, h_K. Figure: 3.1 shows the iterative framework architecture.

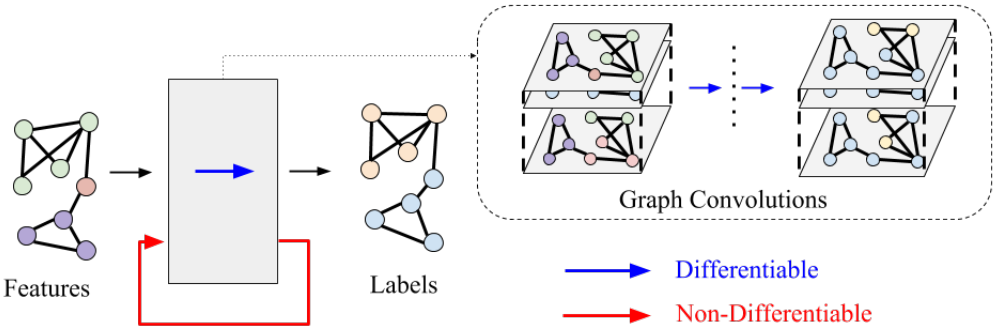


Figure 3.1: GCNs coupled with iterative learning

We explain HOPF's mechanism with a toy chain graph illustrated in Fig: 3.2. The graph has 6 nodes with attributes ranging over A-F and the graph kernel used is of the second order. The figure is intended to explain how differentiable and non-differentiable layers are interleaved to allow propagation up to the diameter. We first analyze it with respect to node 1. In the first iteration, node 1 has learned to aggregate attributes from node 2 and 3, viz *BC*, along with its own. This provides it with an aggregate of information from A, B and C. At the start of each subsequent iteration, label predictions are made for all the nodes using a K^{th}(In Fig: 3.2, $K = 2$) order differentiable kernel learned in the previous iteration. These labels are concatenated with node attributes to form the features for the current iteration. By treating the labels as non-differentible entities, we stop the gradients from propagating to the previous iteration and hence the model is only $K = 2$ hop differentiable.

With the concatenated label information, the model can be made to re-learn from scratch or continue on top of the pre-trained model from the last iteration. Following this setup, one can observe that the information of nodes D, E, and F which is not accessible with a 2^{nd} order differentiable kernel(blue paths) is now accessible via the non-differentiable paths (red and green paths). In the second iteration, information from nodes at 3^{rd} and 4^{th} hop (D and E) becomes available and in the subsequent iteration, information from the 5^{th} hop (F) becomes available. The paths encoded in blue, purple

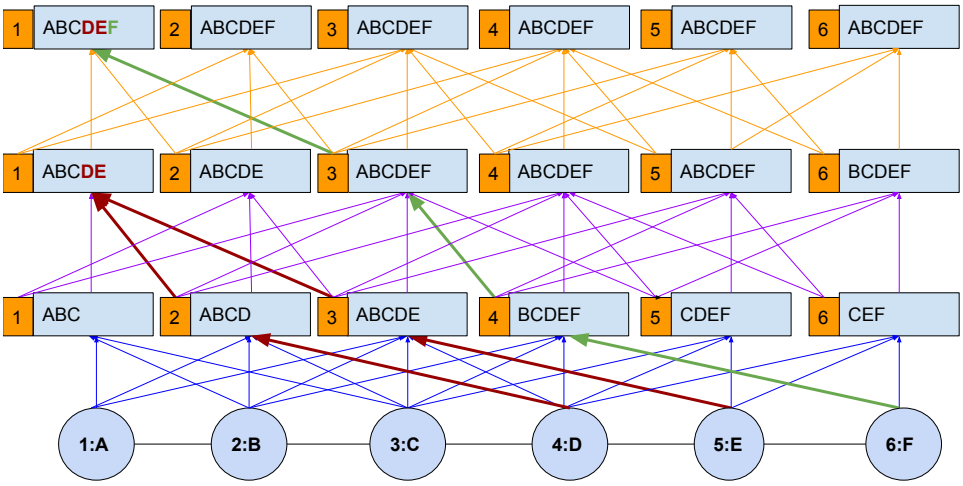


Figure 3.2: HOPF explained with a chain graph

and orange represent different iterations in the figure and are differentiable only during their ongoing iteration, not as a whole.

3.5 Iterative NIP Mean Kernel: I-NIP-MEAN

In this section, we propose a special instance of HOPF which addresses the NIM issue with NIP kernels in a scalable fashion. Specifically, we consider the following NIP Kernel instantiation, *I-NIP-MEAN* with mean aggregation function, by setting $F(A) = D^{-1}A$, $\Phi_k = h_0$, $\Psi = h_{k-1}$, $\Xi^{t-1} = \hat{Y}^{t-1}$ and $W_K^{\phi} \neq W_K^{\psi}$.

$$
\begin{aligned}
h_0^0 &= X; \quad \hat{Y}^0 = 0 \\
h_k^t &= \sigma(\boldsymbol{h_0^t} W_k^{\phi} + D^{-1}A[h_{k-1}^t, \boldsymbol{\hat{Y}^{t-1}}]W_k^{\psi})
\end{aligned} \tag{3.10}
$$

Algorithm 1 describes the algorithm for I-NIP-MEAN model. The iterative learning and inference steps are described in *lines: 7-10* and *12-16* respectively. Both learning and inference happen in mini-batches, $nodes$, sampled from the labeled set, S or the unlabeled set, U respectively as shown in $lines : 8$ and 12 correspondingly. The *predict* function described in *lines:17-27* is used during learning and inference to obtain label predictions for nodes, $nodes$. The procedure first extracts K-hop relational features (h_K) and then projects it to the label space and applies a sigmoid or a softmax depending on the task, see *line: 27*.

Algorithm 1: I-NIP-MEAN: Iterative NIP Mean Kernel

1 ***Input:*** $Dataset : (G, S, U, X, Y)$,
2 No: differentiable hops: C, No: of iterations: T
3 ***Output:*** $\hat{Y}$
4 $\hat{Y}[S] = 0$; $\hat{Y}[U] = 0$; $\tilde{Y} = \hat{Y}$
5 **for** t *in 1:T* **do**
6 // Learning
7 **for** *epoch_id in 1:Max_Epochs* **do**
8 **for** $nodes$ *in* S **do**
9 $\tilde{Y}[nodes]$ = predict$(nodes, G, X, \hat{Y}, K)$
10 min Loss$(\tilde{Y}[nodes], Y[nodes])$
11 // Inference
12 **for** $nodes$ *in* U **do**
13 $\tilde{Y}[nodes]$ = predict$(nodes, G, X, \hat{Y}, K)$
14 $\hat{Y}[S] = Y$
15 // Temporal averaging of predicted labels
16 $\hat{Y}[U] = (T - t)/T * \tilde{Y} + (t/T) * \hat{Y}[U]$
17 **Function** *predict*$(nodes, G, X, \hat{Y}, K)$
18 $A, nodes^*$ = get_subgraph$(G, nodes, K)$
19 $X = X[nodes^*]$; $\hat{Y} = \hat{Y}[nodes^*]$
20 // Compute 0-hop features
21 $h_0 = \sigma(XW_0)$
22 // Compute K-hop features
23 **for** k *in* $1 : K$ **do**
24 $h_k = \sigma(\alpha[\boldsymbol{h_0}]W_k^{\phi} + \beta F(A, [\boldsymbol{h_{k-1}}, \hat{\boldsymbol{Y}}]W_k^{\psi}))$
25 // Predict labels
26 $\tilde{Y} = \sigma(h_K[nodes]W_L)$
27 **return** $\tilde{Y}$

To extract K-hop relational features for $nodes$, the model via *get_subgraph* function first gathers all $nodes$ along with their neighbors reachable by less than $K + 1$ hops ($nodes*$) and represents this entire sub graph by an adjacency matrix (A). A K-hop representation is then obtained with the kernel as in *lines:21-24*. At each learning phase, the weights of the kernels (W_k^{ϕ}s and W_k^{ψ}, $\forall k$) are updated via back-propagation to minimize an appropriate loss function.

3.6 Scalability analysis:

In most real-world graphs exhibiting power law, the size of the neighborhood for each node grows exponentially with the depth of neighborhood being considered. Storing all the node attributes, the edges of the graph, intermediate activations, and all the associated parameters become a critical bottleneck. Here we analyze the efficiency of proposed work to scale to large graphs in terms of the reduction in the number of parameters and space and time complexity.

Number of parameters: The ratio of available labeled nodes to the unlabeled nodes in a graph is often very small. As observed in Kipf and Welling (2016) and Hamilton *et al.* (2017), the model tends to easily over-fit and perform poorly during test time when additional parameters (layers) are introduced to capture deeper neighborhood. In our proposed framework with iterative learning and inference, the parameters of the kernel at $(t-1)^{th}$ iteration is used to initialize t^{th} kernel and is then discarded, hence the model parameters is $O(C)$ and not $O(K)$. Thus the model can obtain information from any arbitrary hop, K with constant learnable parameters of $O(C)$, where $C = T/K$. But in the inductive setup, the parameter complexity is similar to GCN and GraphSAGE as the kernel parameters from all iterations are required to make predictions for unseen nodes.

Space and Time complexity: For a Graph $G = (V, E)$, we consider aggregating information up to K hop neighborhood. Let number of nodes $N = |V|$, and average degree $p = 2|E|/N$. For making full batch updates over the graph (like in GCN), computational complexity for information aggregation is $O(NpK)$, and memory required is $O(NK + |E|)$. Even for moderate size graphs, dealing with such memory requirement quickly becomes impractical. Updating parameters in mini-batches trades off memory requirements with computation time. If batches of size b (where, $0 < b/N << 1$) are considered, memory requirement reduces but in worst case, computation complexity increases exponentially to $O(Np^K)$ as neighborhood of size $O(bp^K)$ needs to be aggregated for each of N/b batches independently. In a highly connected graph (such as a PPI, Reddit, Blog etc.), the neighborhood set of a small K may already be the whole network, making the task computationally expensive and often infeasible. To make this tractable, GraphSAGE considers a partial neighborhood information. Though useful, the use of neighborhood information can also significantly hurt the performance in cer-

tain cases as shown on citation networks, Cora (Lu and Getoor, 2003) and Cora2 (McDaid *et al.*, 2011) in Figure: 3.3. Not only it hurts the performance but it also requires additional hyperparameter tuning for neighborhood size. In comparison, *the proposed work reduces complexity from exponential to linear* $O(NTp^C)$ in the total number of hops considered, by doing T iterations of a constant C hop differentiable kernel, such that $T \times C = K$. In our experiments, we found that even C as small as 2 and $T = 5$ was sufficient to outperform existing methods on most of the datasets. The best models were the ones whose C was the largest hop which gave the best performance for the differnetiable kernel.

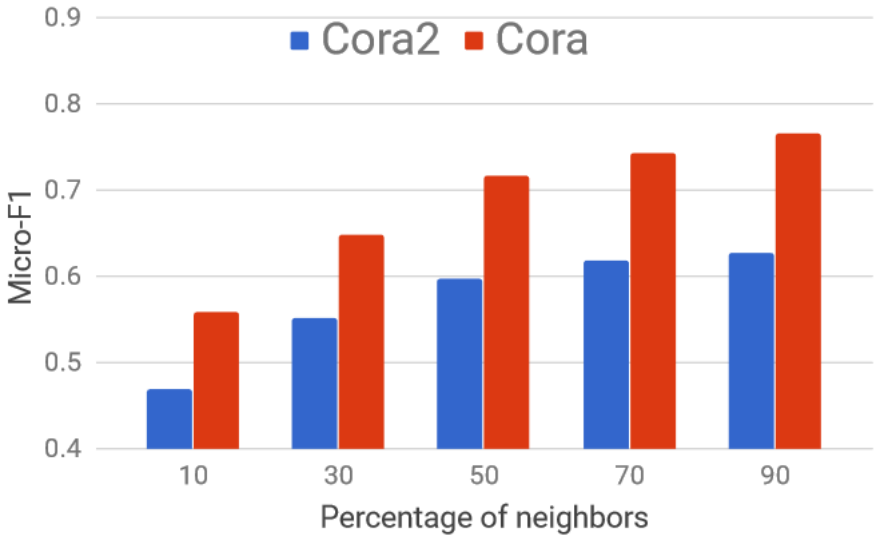


Figure 3.3: Impact of NIP-Mean's performance as percentage of neighbors considered

3.7 Miscellaneous related works

Many extensions of classical methods have been proposed to capture higher-order relational properties of the data. Glocalized kernels (Morris *et al.*, 2017) are a variant of the k-dimensional Weisfeiler-Lehman (Weisfeiler and Lehman, 1968) kernel for graph level tasks that use a stochastic approximation to aggregate information from distant nodes. The differentiable kernels are all 1-dim WL-Kernels whose direct adaptation suffers from Node Information Morphing. Relational classifier (Macskassy and Provost, 2003) builds upon the homophily assumption in the graph structure and diffuses the available label data to predict the labels of unlabelled ones. To make this process more efficient, propagation kernels (Neumann *et al.*, 2016) provide additional schemes for diffusing

the available information across the graph. However, none of these provide a mechanism to adapt to the dataset by learning the aggregation filter.

From a dynamical systems perspective, predictive state representations (Sun *et al.*, 2016) also make use of iterative refinement of internal representations of the model for sequential modeling tasks. However, no extension to graph models has been mentioned. In computer vision application, iterative Markov random fields (Subbanna *et al.*, 2014; Yu and Clausi, 2005) have also been shown to be useful for incrementally using the local structure for capturing global statistics. In this work, we restrict our focus to address the limitations of the current state-of-the-art differentiable graph kernels to provide higher order information for the Collective Classification task. Moreover, HOPF additionally leverages label information that is found useful.

Message Passing Neural Network (Gilmer *et al.*, 2017) is a message passing framework which contains the message and read out component. They are defined for graph level tasks. HOPF is explicitly defined for node level tasks and aims at scaling existing graph networks. HOPF's generic propagation kernel is more detailed than MPNN's message component and can additionally support iterative learning and inference.

3.8 Experiments

In this section, we describe the datasets used for our experiments, the experimental setup, the implementation details and the models compared.

3.8.1 Dataset details

We extensive evaluate the proposed models and the baselines on multiple datasets from various domains. In this work, we treat these networks as undirected graphs but the proposed framework can also handle directed graphs with non-negative edges. The statistics of datasets used are provided in Table: 3.2 and the details are described below:

Social networks: We use Facebook (FB) from Pfeiffer III *et al.* (2015); Moore and Neville (2017), BlogCatalog (BLOG) from Wang *et al.* (2010) and Reddit dataset from Hamilton *et al.* (2017). In the Facebook dataset, the nodes are Facebook users

Table 3.2: Dataset stats: |V|, |E|, |F|, |L|, L_m denote number of nodes, edges, features, labels and is it a multi-label dataset ?

Dataset	Network	|V|	|E|	|F|	|L|	L_m
Cora	Citation	2708	5429	1433	7	F
Citeseer	Citation	3312	4715	3703	6	F
Cora2	Citation	11881	34648	9568	79	T
Pubmed	Citation	19717	44327	500	3	F
Yeast	Biology	1240	1674	831	13	T
Human	Biology	56944	1612348	50	121	T
Reddit	Social	232965	5376619	602	41	T
Blog	Social	69814	2810844	5413	46	T
Fb	Social	6302	73374	2	2	F
Amazon	Product	16553	76981	30	2	F
Movie	Movie	7155	388404	5297	20	T

and the task is to predict the political views of a user given the gender and religious view of the user as features. In the BlogCatalog dataset, the nodes are users of a social blog directory, the user's blog tags are treated as node features and edges correspond to friendship or fan following. The task here is to predict the interests of users. In Reddit, the nodes are the Reddit posts, the features are the averaged glove embeddings of text content in the post and edges are created between posts if the same users comment on both. The task here is to predict the sub-Reddit community to which the post belongs.

Citation Networks: We use four citation graphs: Cora (Lu and Getoor, 2003), Citeseer (Bhattacharya and Getoor, 2007), Pubmed (Namata *et al.*, 2012) and Cora-2 (Mccallum, 2001). In all the four datasets, the articles are the nodes and the edges denote citations. The bag-of-word representation of the article is used as node attributes. The task is to predict the research area of the article. Apart from Cora-2, which is a multi-label classification dataset from Mccallum (2001), others are multi-class datasets.

Biological networks: We use two protein-protein interaction network: Yeast and Human. Yeast dataset is part of the KDD cup 2001 challenge (Hatzis and Page, 2001) which contain interactions between proteins. The task is to predict the function of these genes. Similarly, the Human dataset, introduced in Hamilton *et al.* (2017), is a protein-protein interaction (PPI) network from Human Tissues. The dataset contains PPI from

24 human tissues and the task is to predict the gene's functional ontology. Features consist of positional gene sets, motif gene sets, and immunology signatures.

Movie network: We constructed a movie network from Movielens-2k dataset available as a part of HetRec 2011 workshop (Cantador *et al.*, 2011). The dataset is an extension of the MovieLens10M dataset with additional movie tags. The nodes are the movies and edges are created between movies if they share a common actor or director. The movie tags form the movie features. The task here is to predict all possible genres of the movies.

Product network: We constructed an Amazon DVD co-purchase network which is a subset of Amazon_060 co-purchase data by Leskovec and Sosič (2016). The network construction procedure is similar to the one created in Moore and Neville (2017). The nodes correspond to DVDs and edges are constructed if two DVDs are co-purchased. The DVD genres are treated as DVD features. The task here is to predict whether a DVD will have Amazon sales rank ≤ 7500 or not.

3.8.2 Experiment setup:

In Table: 3.4, we report the averaged test results for transductive experiments obtained from models trained on the 5 different training sets. We also report results on the Transfer (Inductive) learning task introduced in Hamilton *et al.* (2017) under their same setting, where the task is to classify proteins in new human tissues (graphs) which are unseen during training. Results on the inductive setup is provided in Table: 3.5.

The experiments follow a SSL setting with only 10% labeled data. We consider 20% of nodes in the graph as test nodes and randomly create 5 sets of training data by sampling 10% of the nodes from the remaining graph. Further, 20% of these training nodes are used as the validation set. We do not use the validation set for (re)training.

Weighted Cross Entropy Loss (WCE)

Models in previous works (Yang *et al.*, 2016*b*; Kipf and Welling, 2016), were trained with a balanced labeled set i.e equal number of samples for each label is provided for training. Such assumptions on the availability of training samples and similar label

distribution at test time are unrealistic in most scenarios. To test the robustness of CC models in a more realistic set-up, we consider training datasets created by drawing random subsets of nodes from the full ground truth data.

It is highly likely that randomly drawn training samples will suffer from severe class imbalance. This Imbalance in class distribution can make the weight updates skewed towards the dominant labels during training. To overcome this problem, we generalize the weighted cross entropy defined in Moore and Neville (2017) to incorporate both multi-class and multi-label setting. We use this as the loss function for all the methods including baselines. The weight ω for the label i is given in the equation below, where $|L|$ is the total number of labels and N_j represents the number of training samples with label j. The weight of each label ω_i is inversely proportional to the number of samples having that label.

$$\omega_i = \frac{\sum_{j=1}^{|L|} N_j}{|L| \times N_i} \tag{3.11}$$

3.8.3 Implementation details

For optimal performance, HOPF models are processed in mini-batches and also make use of queues to pre-fetch the exponential neighborhood information of nodes of a mini-batch. The recursive propagation steps are computed with sparse-dense computations. The processed data and code would be made publicly available soon. Mini-batching also makes it possible to efficiently distribute the gradient computation in a multi-GPU setup, we leave this enhancement for future work. The choice of data structure for the kernel is also crucial for processing the graph, i.e trade-off between adjacency list and adjacency matrix results. Working with maxpool or LSTMs are difficult using adjacency matrix as the node's neighborhood information needs to be flattened dynamically. Models based on LSTM will also have to deal with issues of nodes having highly varying degrees and limitations of sequential processing of nodes even at the same hop distance. The code for HOPF framework processes neighborhood information with adjacency matrices and is primarily suited for weighted mean kernels.

Hyper-parameters

The hyper-parameters for the models are the number of layers of neural network (hops), dimensions of the layers, dropouts for all layers and L2 regularization, similar to Kipf and Welling (2016). We train all the models for a maximum of 2000 epochs using Adam (Kingma and Ba, 2014) with the initial learning rate set to 1e-2. We use a variant of patience method with learning rate annealing for early stopping of the model. Specifically, we train the model for a minimum of 50 epochs and start with a patience of 30 epochs and drop the learning rate and patience by half when the patience runs out (i.e when the validation loss does not reduce within the patience window). We stop the training when the model consecutively loses patience for 2 turns. We added all these components to the baseline codes too. In fact, we observed an improvement of 25.91 percentage for GraphSage on their dataset. GraphSAGE's LSTM model gave Out of Memory error for Blog, Movielens, and Cora2 as the initial feature size was large and with the large number parameters for the LSTM model the parameter size exploded. Hence, for these datasets alone we reduced the features size and ran the LSTMs.

We searched for optimal hyper-parameter setting on a two-layer deep feedforward neural network with the node attributes (BL_NODE) alone. We then use the same hyper-parameters across all the other models. We row-normalize the node features and use Glorot initialization (Glorot and Bengio, 2010) for weights. Since the percentage of different labels in training samples can be significantly skewed, similar to Moore and Neville (2017) we weigh the loss for each label inversely proportional to its total fraction as in Eqn: 3.11. We ensure that all models have the same setup in terms of the weighted cross entropy loss, the number of layers, dimensions, patience based stopping criteria and dropouts. The best results for models across multiple differentiable hops were reported, 3 hops for Amazon, 4 hops for Cora2 and HUMAN and 2 hops for the remaining datasets. The number of outer steps T for iterative learning was fixed to 5. For the Reddit dataset, we used partial neighbors 25 and 10 in 1^{st} and 2^{nd} hop which is the default GraphSAGE setting as the dataset had extremely high link density. Methods are compared and evaluated on Micro-F1 metric.

Table 3.3: Hyperparameters for different datasets

Hyperparams	CORA	CITE	CORA2	YEAST	HUMAN	BLOG	FB	AMAZON	MOVIE	Pubmed	Reddit
Learning Rate	1E-02	1E-02	1E-02	1E-02	1E-02	1E-02	1E-02	1E-02	1E-02	1E-02	1E-02
Batch Size	128	128	128	128	512	512	128	512	64	128	512
Dimensions	16	16	128	128	128	128	8	8	128	16	128
L2 weight	1E-03	1E-03	1E-06	1E-6	0	1E-06	0	0	1E-06	1E-3	0
Dropouts	0.5	0.5	0.25	0.25	0	0	0	0	0	0.5	0
WCE	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	No
Activation	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU

3.8.4 Models compared:

We compare the proposed NIP models with various differentiable WL kernels, Semi-Supervised ICA and two baselines, BL_NODE and BL_NEIGH as defined in Table: 3.1. BL_NODE is a K-layer feedforward network that only considers the node's information ignoring the relational information whereas BL_NEIGH ignores the node's information and considers the neighbors' information. BL_NEIGH is a powerful baseline which we introduce. It is helpful to understand the usefulness of relational information in datasets. In cases where BL_NEIGH performs poorer than BL_NODE, the dataset has less or no useful relational information to extract with the available labeled data and vice versa. In such datasets, we observe no significant gain in considering beyond one or two hops. All the models in Table: 3.4 and Table: 3.5 except SS-ICA, GCN and GraphSAGE models have skip connections. GraphSAGE models combine node and neighborhood information by concatenation instead of summation.

3.9 Results and Discussions

In this section, we first provide measures for evaluating the consistency of models across datasets and then provide results of our experiments as summarized in Tables 3.4 and 3.5 followed by analysis on the same.

3.9.1 A measure of consistency across datasets

When comparing multiple algorithms across multiple datasets, one naive way of identifying the best algorithm is to simply compare the number of datasets on which each algorithm gives the best results. For example, it is evident from Table 3.4, that the proposed algorithm I-NIP-MEAN gives the best results on 5/11 datasets followed by

SS-ICA which gives the best results on 2/11 datasets. By this naive measure of counting the number of wins of a given algorithm, the proposed method outperforms existing methods. However, we argue that this is not the correct way of comparing multiple algorithms on different datasets. For example, there could be an algorithm which is consistently the second best algorithm on all the datasets with minute difference from the best and yet have zero wins. To capture this notion of consistency, we introduce a measure, *shortfall*, which captures the relative shortfall in performance compared to the best performing model on a given dataset.

$$shortfall[data] = \frac{best[data] \text{ - } performance[data]}{best[data]} \tag{3.12}$$

Where *best[dataset]* is the micro_f1 of the best performing model for the dataset and *performance[data]* is the model's performance for that dataset. In Table: 3.4, we report the average *shortfall* across datasets. Lower *shortfall* indicates a better consistent performance. Even using this measure the proposed algorithm I-NIP outperforms existing methods. In particular, notice that while SS-ICA seemed to be the second best algorithm usig the naive method of counting the number of wins, it does very poor when we consider the *shortfall* metric. This is because SS-ICA is not consistent across datasets and in particular it gives a very poor performance on some datasets which is undesirable. On the other hand, I-NIP-MEAN not only wins on 5/11 datasets but also does consistently well on all the datasets and hence has the lowest average *shortfall*.

Table 3.4: Results in Micro-F1 for Transductive experiments. Lower *shortfall* is better. Top two results in each column in bold.

	Datasets											Aggregate measures	
MODELS	Blog	FB	Movie	Cora	Citeseer	Cora2	Pubmed	Yeast	Human	Reddit	Amazon	Shortfall	Rank
BL_NODE	37.929	**64.683**	50.329	59.852	65.196	40.583	83.682	59.681	41.111	57.118	64.121	16.9	8.82
BL_NEIGH	19.746	51.413	35.601	77.43	70.181	63.862	83.16	53.522	60.939	59.699	66.236	17.3	8.45
GCN	34.068	50.397	39.059	76.969	**72.991**	**63.956**	85.722	62.565	58.298	75.667	61.777	11.0	6.64
GCN-S	39.101	63.682	51.194	**77.523**	71.903	63.152	**86.432**	60.34	62.057	77.637	**73.746**	4.1	4.36
GCN-MEAN	38.541	62.651	51.143	76.081	**72.357**	62.842	85.792	61.787	64.662	74.324	63.674	5.6	6
GS-MEAN	**39.433**	64.127	50.557	76.821	70.967	62.8	84.23	59.771	63.753	79.051	68.266	4.9	6
GS-MAX	**40.275**	64.571	50.569	73.272	71.39	53.476	85.087	62.727	65.068	78.203	70.302	5.5	4.73
GS-LSTM	37.744	**64.619**	41.261	65.73	63.788	38.617	82.577	58.353	64.231	63.169	68.024	14.4	8.45
NIP-MEAN	**39.433**	64.286	51.316	76.932	71.148	63.901	**86.203**	61.583	**68.688**	77.262	69.136	3.6	4
SS-ICA	38.517	64.349	**52.433**	75.342	68.973	63.098	84.798	**68.444**	43.629	**81.92**	65.789	6.6	5.73
I-NIP-MEAN	39.398	62.889	**51.864**	**78.854**	71.541	**66.23**	85.341	**69.917**	**68.652**	**81.64**	**75.045**	**0.9**	**2.81**

Table 3.5: Results in Micro-F1 for Inductive learning on Human Tissues

	Node	Neighbor	NIP-MEAN	GCN-MEAN	GCN	GS-Mean	GS-Max	GS-LSTM	SS-ICA	I-NIP-MEAN
PPI	44.51	83.891	**92.243**	86.049	88.585	79.634	78.054	87.111	61.51	**92.477**

3.9.2 Baselines Vs. Collective Classification (CC) models

As mentioned earlier, the baselines BL_NEIGH and BL_NODE use *only neighbor* and *only node* information respectively. *In datasets, where BL_NEIGH significantly outperform BL_NODE, all CC models ouperform both these baselines by jointly utilizing the node and neighborhood information.* In datasets such as Cora, Citeseer, Cora2, Pubmed and Human, where performance of BL_NEIGH $>$ BL_NODE, CC models improve over BL_NEIGH by up to 8% in the transductive setup. Similarly, on the inductive task where the performance of BL_NEIGH is greater than BL_NODE by $\approx 40\%$, CC methods end up further improving by another 8%. In Reddit and Amazon datasets, where the performance of BL_NODE $\approx$ BL_NEIGH, CC Methods still learn to exploit useful correlations between them to obtain a further improvement of $\approx 20\%$ and $\approx 10\%$ respectively.

3.9.3 WL-Kernels Vs NIP-Kernels

We make the following observations based on the performance of WL-Kernels and NIP-Kernels. First, we empirically observe the issue of Node Information Morphing in WL-kernels. Then, we provide empirical evidence suggesting the benefits of using skip connection followed by highlighting the superiority of NIP kernels over WL-Kernels.

1. Node Information Morphing in WL-Kernels: The poor performance of BL_NEIGH compared to BL_NODE on the Blog, FB and Movie datasets suggests that the neighborhood information is noisy and node features are more crucial. *The original GCN which aggregates information from the neighbors but does not use CONCAT or skip connections typically suffers a severe drop in performance of up to $\approx 13\%$ on datasets with high degree.* Despite having the node information, GCN performs worse than BL_NODE on these datasets.

2. Solving NIM with skip connections in WL-Kernels: The original GCN architecture does not allow for skip connections from h_0 to h_1 and from h_{K-1} to h_K. We modify

the original architecture and introduce these skip connections (GCN-S) by extracting h_0 features from the 1^{st} convolution's node information. *With skip connections, GCN-S outperforms the base GCN on 8/11 datasets.* The performance boost is $\approx 5 - 13\%$ in the case of Blog, FB, Movie and Amazon datasets even when we consider only 2 hops thereby decreasing the *shortfall* on these datasets. In particular, the issue of NIM is resolved as the performance of GCN-S now comes close to that of BL_NODE and in the case of Amazon dataset it actually outperformcs BL_NODE. GCN-MEAN which also has skip connections performs quite similarly to GCN-S in all datasets and does not suffer from NIM as much as GCN. *It is important to note that skip connections are required not only for going deeper but more importantly, to avoid information morphing even for smaller hops.*

GS models do not suffer from NIM issue as they concatenate node and neighborhood information. Authors of GS also observed better results in their tasks with the addition of CONCAT. GS-MEAN's counterpart among the summation models is the GCN-MEAN model which gives similar performance on most datasets, except for Reddit and Amazon where GS-MEAN with concat performs better than GCN-MEAN by $\approx 5\%$. GS-MAX provides very similar performances to GS-MEAN, GCN-MEAN, and GCN-S across the board. Their shortfall performances are also very similar. GS-LSTM typically performs poorly which might be because of the morphing of earlier neighbors' information by more recent neighbors by in the list.

3. Solving NIM with NIP Kernels: *NIP-MEAN, a MEAN pooling kernel from the NIP propagation family outperforms its WL family counterpart, GCN-MEAN on 9/11 datasets.* It achieves a significant improvement of $\approx 3 - 6\%$ over GCN-MEAN in Human, Reddit and Amazon datasets. It similarly outperforms GS-MEAN on another 9/11 datasets even though GS-MEAN has twice the number of parameters. NIP-MEAN provides the most consistent performance among the non-iterative models with a shortfall as low as 3.6. NIP-MEAN's clear improvement over its WL-counterparts demonstrates the benefit of using NIP family of kernels which explicitly preserve the node information and mitigate the NIM issue.

3.9.4 Iterative inference models Vs. Differentiable kernels

Iterative inference models, SS-ICA and I-NIP-MEAN exploit label information from the neighborhood and scale beyond the memory limits of differentiable kernels. The benefit of label information over attributes can be analyzed with SS-ICA which aggregates only the label information of immediate neighbors. In Yeast dataset, SS-ICA gains $\approx 8 - 10\%$ improvement over non-iterative models which do not use label information. Similarly, in Reddit dataset, both attribute and label information are equally useful while using label information seems to provide slightly better results as SS-ICA manages to obtain 81.92 starting from 57.118 (BL_NODE). However, SS-ICA does not give good performance on some datasets as it does not leverage neighbors features and is restricted to only learn first-order local information unlike multi-hop differentiable NIP and other WL-based kernels.

Iterative Differentiable kernels Vs. Rest

I-NIP-MEAN which is an extension of NIP-MEAN with iterative learning and inference can leverage attribute information and exploit non-linear correlations between the labels and attributes from different hops. *I-NIP-MEAN improves over NIP-MEAN on 8/11 datasets with significant boost in performance up to* $\approx 3 - 8\%$ *in cora2, Reddit, Amazon, and Yeast datasets.* I-NIP-MEAN also successfully leverages label information like SS-ICA and obtains similar performance boost on Yeast and Reddit dataset. *It also outperforms SS-ICA on 8/11 datasets.* The benefits of using neighbors' attributes along with labels are visible in Amazon and Human datasets where I-NIP-MEAN model achieves $\approx 10\%$ and $\approx 25\%$ improvement respectively over SS-ICA which uses label information alone. I-NIP-MEAN combines the best of both the worlds, viz., differentiable kernel based methods and iterative inference based methods and *emerges as the most robust model across all datasets with the lowest shortfall of* $\approx 0.9\%$.

3.9.5 Inductive learning on Human dataset

For the inductive learning task in Table: 3.5, the cc models obtain a 44% improvement over BL_NODE by leveraging relational information. The I-NIP-MEAN and NIP-MEAN kernels achieves best performance with a $\approx 6\%$ improvement over GCN-MEAN by explicitly capturing the correlations between the node and multi-hop neighborhood information.

CHAPTER 4

Fusion Graph Convolutional Networks

Summarizing information from multiple hops is useful in many applications where there exists semantics in local and group level interactions among entities. Thus, defining and finding the significance of neighborhood information over multiple hops becomes an important aspect of the problem. However, on analysis we find that existing WL-kernel based GCNs and GraphSAGE models have a restricted formulation that hinders them from effectively learning relevant relational features.

In this chapter, we first provide a simplified re-formulation of the generic kernel in 3.1 aimed at abstracting GCN and GraphSAGE models alone. Then, we analyze the representation capacity of these models to regulate information from multiple hops independently. From our analysis, we show that these models despite being powerful, have limited representation capacity to capture multi-hop neighborhood information effectively. Further, we also propose a mathematically motivated, yet simple extension to existing GCNs which has improved representation capacity. Finally, we report experimental evidences supporting the analysis and proposed model.

4.1 Unified Recursive Graph Propagation Kernel

GCN and GraphSAGE differ in terms of the neighborhood features considered (Ψ_k), node's importance (α) and their method of combining node and neighborhood information.GCNs have shared weights for node and neighbors features and it combines these information by summation. Whereas, GraphSAGE combines information by concatenation ($CONCAT$) and hence will have different weights for them.

Focused only on these differences and additionally on the mode of combination of information, we simplify and re-formulate the generic kernel in Eqn: 3.1 as below:

$$\begin{aligned} h_k &= \sigma_k(combine(\Phi_k, \Psi_k)W_k) \\ \Phi_k &= \alpha h_{k-1} \\ \Psi_k &= F(A)h_{k-1} \end{aligned} \tag{4.1}$$

Where Φ_k and Ψ_k denote the $(k-1)^{\text{th}}$ hop node and it's neighbors' features respectively, α denotes the scaling factor for node features, $F(A)$ denotes the neighborhood aggregation function, and $combine$ denotes the mode of combination of node and it's neighbors' features. For brevity, we have made the neighbors' weighting function to be independent of h_{k-1}.

The concatenation combination (denoted by square braces below) can also be expressed in terms of a summation of node and neighbors features with different weight matrices, $W_k^{\phi} \in \mathbb{R}^{d \times d}$ and $W_k^{\psi}\mathbb{R}^{d \times d}$ respectively by appropriately padding zero matrices, ($\mathbf{0}$) as shown below.

$$\begin{aligned} h_k &= \sigma_k([\alpha h_{k-1}, F(A)h_{k-1}][W_k^{\phi}, W_k^{\psi}]) \\ h_k &= \sigma_k([\alpha \cdot h_{k-1}, \mathbf{0}][W_k^{\phi}, \mathbf{0}] + [\mathbf{0}, F(A)h_{k-1}][\mathbf{0}, W_k^{\psi}]) \\ h_k &= \sigma_k([\alpha \cdot h_{k-1}W_k^{\phi}, \mathbf{0}] + [\mathbf{0}, F(A)h_{k-1}W_k^{\psi}]) \end{aligned} \tag{4.2}$$

The Φ_k and Ψ_k terms for CONCAT and SUMMATION combinations are similar if weights are shared in the CONCAT formulation as shown in Eqn: 4.3 and Eqn: 4.4 respectively, i.e $W_k^{\phi} = W_k^{\psi} = W_k$.

$$h_k = \sigma_k(([\alpha h_{k-1}, 0] + [0, F(A)h_{k-1}])[W_k, W_k]) \tag{4.3}$$

$$h_k = \sigma_k(\alpha h_{k-1} + F(A)h_{k-1}W_k) \tag{4.4}$$

For brevity of analysis made henceforth, we only consider the summation model to discuss the limitations of the recursive propagation kernels without losing any generality on the deductions made. Further, we provide another abstraction to the summation formulation as in Eqn: 4.4 by Eqn: 4.5. Henceforth, we refer to Eqn: 4.5 as the generic recursive propagation kernel in the upcoming analysis.

$$\begin{aligned} \Omega &= (\alpha + F(A)) \\ h_k &= \sigma_k(\Omega h_{k-1}W_k) \end{aligned} \tag{4.5}$$

4.2 Lack of independent regulatory paths to different hops

Neighbor's from different hops might contribute differently to the end classification task depending on the problem and the network dataset. Depending on the task, the influence of a neighbor on a node might be constant or increase or decrease with an increase in the neighborhood depth, K. There can be an alternating influence too, consider a simple predator-prey network where the two kinds of information can be captured at alternate hops. In general, networks can naturally have different k-partite structure locally or globally; therefore information from a specific partition might need to be ignored or grouped separately to find relevant nodes. In the absence of prior knowledge, this is hard to hand-code for large graphs. Hence it is necessary for neural propagation models to have the flexibility to adapt and learn the importance of different neighborhood information accordingly.

However, though these propagation models can combine information from multiple hops, their formulation restricts them from independently regulating information from different hops. This is a consequence of recursively computing K^{th} hop information in terms of $(K\text{-}1)^{\text{th}}$ hop information which results in interdependence among weights associated with the different hop information. We can see this below in the recursively expanded unified graph kernel.

$$h_K = \sigma_K(\Omega \cdot ...\sigma_2(\Omega \cdot (\sigma_1(\Omega \cdot h_0 W_1)W_2)...W_K) \tag{4.6}$$

Let's analyze this with an example of a 3-hop linear kernel with K=3 and σ_k = I which on expansion yields the following equation:

$$\begin{aligned} h_3 = \alpha^3 h_0 \Pi_{k=1}^{k=3} W_k + 3\alpha^2 F(A) h_0 \Pi_{k=1}^{k=3} W_k \\ + 3\alpha F(A)^2 h_0 \Pi_{k=1}^{k=3} W_k + F(A)^3 h_0 \Pi_{k=1}^{k=3} W_k \end{aligned} \tag{4.7}$$

This expansion makes it trivial to note that all the weights influence all the different hop information ($h_0, F(A)h_0, F(A)^2 h_0$ and $F(A)^3 h_0$) in the model. For example, if we take the case where only first-hop information (just $F(A)$ term) is required, then there exists no combination of W_ks that can provide it under the current model. It should be

noted that we cannot obtain the 1^{st} hop information alone by using a 1-hop kernel as that would also include 0-hop information, $F(A)^0 h_0 = h_0$.

From the above analysis, we can say that these recursive graph kernels have *limited representation capacity* as they cannot capture information from a particular subset of hops without including information from other hops. The limitation of these networks can be attributed to the specific formulation of recursion used to compute output at every layer. As with every layer k of graph convolutional nets, a new information about the k^{th} hop is introduced as $\Omega = I + \boldsymbol{F(A)}$ in $h_k = \sigma_k(\Phi h_{k-1} W_k)$ whereas with the conventional feed forward nets there is no inclusion of new information at every layer, k as $\Phi = I$ in $h_k = \sigma(\Phi h_{k-1} W_k)$. More importantly, the output at k^{th} layer passes through a series of computations involving later hops ($j > k$) before reaching the last output layer. And also note that this phenomenon happens for previous layers too. Thus, this leads to a lack of independent computation paths to regulate information from any hop without affecting information from later and earlier hops.

In the remainder of this section, we analyze the representation capacity of these propagation models by incorporating additional popular components as follows.

4.2.1 Inclusion of bias

An inclusion of bias term in the recursive propagation kernel as in Eqn: 4.8 is not useful.

$$h_k = \sigma_k(\Phi \cdot h_{k-1} W_k + b_{k-1}) \tag{4.8}$$

The model still cannot regulate the relative magnitude of information independently as information at each hop, h_k (with the bias) is still computed recursively in terms of the previous hop. The resultant inter-dependency can be seen in Eqn: 4.9 with expansion of Eqn: 4.8. This was also observed empirically that the addition of bias did not result in any significant change in results (not reported here).

$$h_K = (\Phi)^K \Pi_{i=1}^K h_0 W_i + \sum_{j=2}^{K} (\Phi)^j (\Pi_{l=k-j+1}^K W_i) b_{k-j} \tag{4.9}$$

4.2.2 Inclusion of skip connections

Adding the popular skip connection (He *et al.*, 2016) to these models as in Eqn: 4.10 improves the multi-hop information regulation capacity.

$$h_k = \sigma_k(\Phi h_{k-1} W_k) + h_{k-1}, \forall k \in [2, K] \tag{4.10}$$

$$h_k = \Sigma_{i=1}^{k} \sigma_k(\Phi \cdot h_{i-1} W_i) \tag{4.11}$$

On recursively expanding the above equation, it can be seen that adding skip connections to a layer, k results in directly adding information from all the lower hops, $i < k$ as shown in Eqn: 4.11. Unlike Eqn: 4.5 where the output at each layer, k was only dependent on the previous layer, h_{k-1} accounting to only one computational path; now adding skip connections allows for multiple computational paths. As it can be seen that at the K^{th} layer the model has the flexibility to select an output from any or all $h_k \forall k < K$.

However, it can only discard information beyond a particular hop, k and is still not sufficient to individually regulate the importance of information from individual hop as all hops, $i \leq k$ are inter-dependent. Lets us consider the same example of a 3-hop model as earlier to capture information from 1^{st} hop alone ignoring the rest. The best, the 3-hop model with skip connections can do is to learn to ignore information from 2^{nd} and 3^{rd} hop by setting W_2=W_3=0 and including h_1 along with h_0. It can be reasoned as before to see that W_0 cannot be set to 0 as h_1 depends on the result of h_0 thereby having no means to ignore information from h_0. This limits the expressive power to efficiently span the entire space of K^{th} order neighborhood information. To summarize, skip connections at best can obtain information up to a particular hop by ignoring information from subsequent hops. The $CONCAT$ combination can be perceived as linear skip connection as noted by the authors of GraphSAGE.

4.2.3 Inclusion of different weights

For convenience, in this subsection alone we change the notations for weights associated with the node and the neighbor features to W_k^0 and W_k^1 instead of W_k^Φ and W_k^Ψ, respectively. The representation capacity of the recursive graph kernel with different

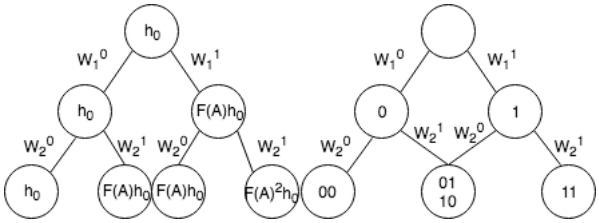


Figure 4.1: Binomial Computation Trees for Graph Kernels

weights for the node and the neighbor features as in Equation 4.12 is better than a kernel with shared weights.

$$h_K = \alpha h_{k-1} W_k^0 + F(A) h_{K-1} W_k^1 \tag{4.12}$$

The flexibility of this formulation can be explained with the binomial expansion as seen earlier in 3.3. At every recursive step, k, the model can be seen deciding to use only the node information or the neighbors' information or both, by manipulating the weights (W_k^0 and W_k^1 to be zero or non-zero values). For convenience, we refer the weights to be set if the weights have non-zero values. The decisions of the model can be better understood when visualized as a binomial tree where the nodes are labeled with the computation output, and the edges are labeled by the decision taken, *i.e.* W_k^0 or W_k^1. The left figure in Figure 4.1 illustrates the computation graph for a 2 hop kernel with different weights.

For any K hop kernel with a K recursive computational layer, there will be 2^K unique paths/decisions to make. The 2^K paths lead to 2^K leaf nodes which compute different $F(A)^k h_0$ terms. $F(A)^k h_0$ terms are available at one or more leaves where the multiplicity of availability is given by the different ways to choose k from K, i.e., $\binom{K}{k}$ (binomial coefficient). h_0 and h_K terms have only one path, i.e $\binom{K}{0} = \binom{K}{K} = 1$, whereas the terms h_k $(0 < k < K)$ have more than one path.

Let string '0' denote the identity transformation, $h_{k-1} W_k^{\mathbf{0}}$ and '1' denote the F(A) transformation, $h_{k-1} W_k^{\mathbf{1}}$. We say a transformation has happened if the weights associated with it have non-zero values. To comprehend the dependencies among weights, let us trace the weights along the different paths to leaf nodes. We create the tree on the right in Figure 4.1 from the output computation tree on the left by relabeling nodes with substrings representing the transformation taken to reach that node. For example, a node labeled '01' indicates that the node was reached by taking an identity transfor-

mation followed by an F(A) transformation. Hence, the number of 0s and 1s at each leaf node conveys the pattern to compute each hop information for a K hop kernel.

In Table 4.1, we tabulate the number of identities, $\#W_k^0 s$ and the number of $F(A)$ transformations, $\#W_k^1 s$ taken to obtain different hop information at the leaves for a 3-hop kernel. $\#Paths$ in the table denote the number of paths to compute the same. With the example in the table, we generalize the following claims to any K hops.

- $F(A)^k$ computation requires $K-k$ identity transformations ($\#W_k^0 s$) and k $F(A)$ transformations ($\#W_k^1 s$).
- All $F(A)^k$ computation has a unique combination of $\#W_k^0 s$ and $\#W_k^1 s$. From this, we can say that the model can learn to obtain any specific hop information without the inclusion of any information from the rest of the hops unlike shared weight models, where all the $F(A)^k$ computations shared the same path.
- We cannot independently regulate information from two or more required hops without the inclusion of information from the others hops lying within the range of the required set. This is a consequence of sharing weights among the computation paths as seen in the Figures.

Leaving out the first two trivial claims we analyze the last claim. Let S define the set of all required hops in a K hop graph kernel. Let i and j be the minimum and the maximum hops in that set. For the i^{th} hop, $(K-i)$ $W_k^0 s$ and i $W_k^1 s$ should be set (non zero values) and similarly for the j^{th} hop $(K-j)$ $W_k^0 s$ and j $W_k^1 s$ should be set. i^{th} and j^{th} hop information can be obtained by traversing any path that would satisfy the previously mentioned conditioned on the number of identity and $F(A)$ transformations. Thus, put together $(K-i)$ $W_k^0 s$ and j $W_k^1 s$ will be set to obtain $F(A)^i h_0$ and $F(A)^j h_0$ when traversed along one path in the computation tree, to the leaf from the root.

Since the model sums up all the leaf nodes, it will also include information from those leaf nodes which can be traversed from the root by following the set W_k^0s and W_k^1s. We go about the proof by first formulating the condition under which other hops' information can be obtained. Then, we go on to show that if $j > i+1$ then $j-i$ additional hop information will be included.

Let y denote hops, with $y \notin \{i, j\}$. Computing the y^{th} hop requires $(K-y)$ $W_k^0 s$ and y $W_k^1 s$. $F(A)^y$ can be obtained only if $C_{K-y}^{K-i} \geq 1$ and $C_y^j \geq 1$, which essentially means that $(K-y)$ $W_k^0 s$ should be a subset of $K-i$ $W_k^0 s$ and y $W_k^1 s$ should be a subset of j $W_k^1 s$ which have already been set while considering i^{th} and j^{th} hop information.

We then find the possible y values under the following three conditions listed below:

	$\#W_k^0 s$	$\#W_k^1 s$	$Paths$
$F(A)^0 h_0$	3	0	1
$F(A)^1 h_0$	2	1	3
$F(A)^2 h_0$	1	2	3
$F(A)^3 h_0$	0	3	1

Table 4.1: Number of Identity and F(A) transformations

- $0 < y < i$: As $y < i < j$, the required number of $W_k^1 s$ for y^{th} hop is available whereas the required number of $W_k^0 s$ are not as $K - 0 > K - i - 1 > K - i$. Hence information from $[0, i)$ is not included.

- $j < y < K$: As $y > j$, the required number of $W_k^1 s$ are unavailable though the required number of $W_k^0 s$ can be satisfied as $K - y < K - j < K - i$.

- $i < y < j$: As $i < y < j$, the required number of $W_k^1 s$ are satisfied and so is the required number of $W_k^0 s$ as $K - i > K - y > K - j$.

This can be clearly seen from the example on the 3^{rd} hop kernel presented in Table 4.1. When the weights along the unique path for 0^{th} and 3^{rd} hop information are set, it sets up all the weights. As 0^{th} hop information is obtained by doing identity transformation at every layer and 3^{rd} hop information is obtained by doing $F(A)$ transformations at every layer, all $W_k^0 s$ and $W_k^1 s$ are set, which would necessarily end up including all the other hop information as all the weights are active. Similarly, it can be shown that when hops 2 and 3 are included, no information from hops 0 and 1 are included.

4.3 Proposed Methodology

In this section, we propose a simple yet effective extension to GCNs by adding a fusion component that allows them to capture multiple hop information effectively. We motivate and propose this component as a solution that will enable these graph kernels to span the entire space of K-hop neighborhood. First, we show that the unified kernel is a binomial combination of node and its' neighborhood information. Thus, at each layer k, a k^{th} hop kernel is computed by a binomial combination. In light of this, we propose a simple fusion layer that learns to linearly combine information from these binomial bases to span the entire K-hop space.

4.3.1 Binomial basis

The K-hop unified propagation kernel defined in Eqn: 4.5 can be rolled out similar to Eqn: 4.6 and be expressed as a K^{th} order binomial in terms of node and it's neighbors' features for the linear activation case as given in Eqn: 4.13.

$$h_K = (\alpha I + F(A))^K h_0 \Pi_{k=1}^{K} W_k \tag{4.13}$$

The higher order binomial term in Eqn: 4.13 when expanded assigns different weights to different $F(A)^k h_0$ terms as seen in Eqn: 4.7. These weights correspond to the binomial coefficients of the binomial series, $(\alpha I + F(A))^K$. For example, refer to Eqns: 4.14 and 4.15 corresponding to a 2-hop and 3-hop kernel with $\alpha = I$ and $W_K = I$ for simplicity. It can be seen that for the 2-hop kernel the weights are $[1, 2, 1]$ and for the 3-hop kernel it is $[1, 3, 3, 1]$. Thus, these recursive propagation kernels combine different hop information weighed by the binomial coefficients.

$$h_2 = h_0 + 2F(A)h_0 + F(A)^2 h_0 \tag{4.14}$$

$$h_3 = h_0 + 3F(A)h_0 + 3F(A)^2 h_0 + 3F(A)^3 h_0 \tag{4.15}$$

These weights induce a bias on the importance of each hop which again is a limitation of the kernel design. Any such fixed bias over different hops cannot consistently provide good performance across numerous datasets. In the limit of infinite data, we can expect the W_k parameters to correct these scaling factors induced by these biases. However, as with most graph-based SSL applications where the amount of available labeled data for training is limited, an undesirable bias can result in a sub-optimal model.

Existing propagation kernels defined over K-hop information, extract relational information by performing convolution operations on different k-hop neighbors based on their respective K^{th} order binomial. As discussed earlier, biasing the importance of information along with recursive weight dependencies hinder the model from learning relevant information from different hops. These limitations constrain the expressive power of these models from spanning the entire space of K^{th} order neighborhood information. Hence, it is restricted to only a subspace of all possible K^{th} order polynomial defined on the neighborhood of nodes.

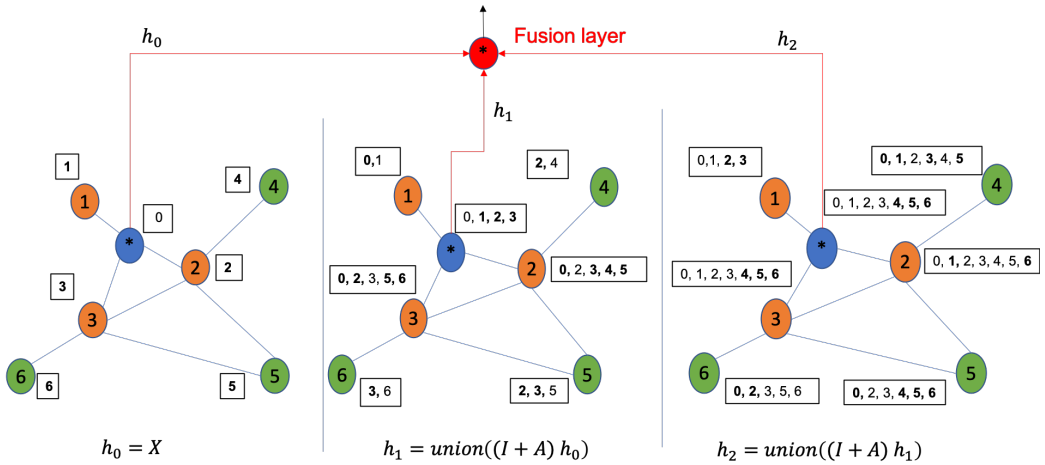


Figure 4.2: Illustration of information propagation in Fusion incorporated WL-kernels

4.3.2 Linear Fusion Component

To mitigate these issues with existing models, we propose a minimalistic component for these models, a **fusion component**. This fusion component consists of parameters to combine the information from the binomial basis defined by the different hop information to effectively scale the entire space of a K^{th} order neighborhood.

We define the fusion component in Eqn: 4.16 as a linear weighted combination over K-hop neighborhood space spanned by the binomial basis, h_ks ($[h_0, \Phi h_0, \ldots, \Phi^K h_0]$) with coefficients, $[\theta_0, \theta_1, \ldots, \theta_K]$. The θ coefficients allow the neural network to explicitly learn the optimal combination of information from different hops. As the h_ks are binomials, a parameterized linear combination of these binomials can obtain any combination of the individual hop information. This can be verified by the fact that the coefficients of the binomial equation which form the Pascal matrix are a non-singular lower triangular matrix. Thus in the linear neural network case, we can see that the binomial system of equations can be solved to obtain any combination of $F(A)$ terms. Continuing the non-differentiable illustration of information propagation in WL-kernels from before, Figure: 4.2 depicts the addition of Fusion component to WL-kernels.

$$y = \Sigma_{k=0}^{K} h_k \theta_k \tag{4.16}$$

4.3.3 Fusion Graph Convolutional Network

We propose Fusion Graph Convolutional Network, F-GCN in equations in 4.17. F-GCN is a minimalistic architecture that adds the fusion component defined in Eqn: 4.16 to GCN defined in Eqn: 2.15. It can be seen to combine different k^{th} hop information with the fusion component. The fusion component mentioned in the penultimate line of the equations in 4.17 fuses label scores from each propagation step. The fused label scores are then normalized to make label prediction.

$$\begin{aligned} h_0 &= \sigma_k(XW_1) \\ h_k &= \sigma_k(\hat{L}h_{k-1}W_k), \ \ \forall\, k \in [1, K]. \\ y &= \Sigma_{k=0}^{K} h_k\theta_k \\ \mathcal{L} &= softmax(y) \ \ or \ \ sigmoid(y) \end{aligned} \tag{4.17}$$

The dimensions of h_k, θ_k, Y , W_0, W_k are $\mathbb{R}^{N\times d}$, $\mathbb{R}^{d\times L}$, $\mathbb{R}^{N\times L}$, $\mathbb{R}^{F\times d}$, $\mathbb{R}^{d\times d}$ respectively. F-GCN uses ReLU activations and a softmax label layer accompanied by a multi-class cross entropy for multi-class classification problem or a sigmoid layer followed by a binary cross entropy layer for multi-label classification problem. Since predictions are obtained from every hop, we also subject h_0 to a non-linear activation function with weights same as W_1 from h_1.

The number of parameters in F-GCN is $O(K * d * d + K * d * L)$, where the first term is for GCN and the second is for the fusion component. GraphSAGE models with no shared weights have $O(K * 2(d * d))$ or $O(K * 2(2d * d))$ for the summation and concat combination respectively which is more than F-GCN as $L < d$ typically. This simple fusion component with fewer parameters provides additional benefits to F-GCN besides explicitly allowing to capture different hop information. It provides for additional direct gradient flow paths to each of the propagation steps allowing it to learn better discriminative features at the lower hops too which also improves its chances of mitigating vanishing gradient. F-GCN can be seen as a multi-resolution architecture which simultaneously looks at information from different resolutions/hops and models the correlations among them.

The fusion component is similar in spirit to the Chebyshev filters introduced in Defferrard *et al.* (2016) for complete graph classification task. The primary difference is

that the Chebyshev filters learn coefficients to combine Chebyshev polynomials defined over neighborhood information whereas in F-GCN the coefficients of the filter are used to combine different binomials pertaining to different layers. Moreover, an additional difference is that the Chebyshev polynomial basis is not associated with weights W_k to filter k^{th} hop information which can potentially enable the model to learn complex non-linear feature basis. F-GCN also enjoys the benefit of the re-normalization trick of GCN that stabilizes the learning to diminish the effect of vanishing or exploding gradient problems associated with training neural networks.

Note that existing attention mechanisms (Vaswani *et al.*, 2017) are typically defined for a positive combination of information. They have a restricted scoring range based on the activation functions used. In our case, since we needed a mechanism that can scale the amount of addition and subtraction of information from different binomial bases, we opted for the simple linear weighted combination layer.

4.4 Relation to other existing works

Being motivated by the problem that the use of the same GCN aggregation function across datasets results in having the same local information retain probability, Xu *et al.* (2018) proposed jump connections that learn the importance of different layers. They use a random walk based analysis to show that with GCNs, neighbors from different hops have the same effect on the node irrespective of network properties. With our unified perspective of different graph models as binomial graph kernels, we can make a broader claim that irrespective of the aggregation function there exists a static bias on the importance of different hops.

In order to overcome this issue, Xu *et al.* (2018) proposed jump connections module with three alternatives viz, CONCAT, MaxPooling and a Gating variant that combines information from different layers. Though Xu *et al.* (2018) shows proof of how a 'learned' Maxpooling or Gating based model corresponds to a mixture of a K-step random walk, they do not provide analysis on the learn-ability of these models. As argued in Xu *et al.* (2018) Maxpooling and Attention layers are more potent than the CONCAT layer as they are adaptive to nodes. However, with our analysis, it can be seen that for binomial graph kernels, CONCAT is more powerful than MaxPooling and Attention

models. Adding a Maxpooling layer to GCN still does not provide a direct independent regulating path to each layer. Though attention function provides a direct regulatory path, the one proposed can only be used for a positive combination of information and is not suited for binomial functions as discussed in the previous section.

Though Xu *et al.* (2018) argued CONCAT based model as weak it achieved superior performance in their experiments. Xu *et al.* (2018)'s CONCAT layer is the same as our linear fusion component [1]. Our well-founded analysis and motivation for the linear fusion component provide a clear rationale and support for the superior experimental benefits obtained with CONCAT/linear fusion both in our work and in Xu *et al.* (2018).

Gilmer *et al.* (2017) proposed Neural Message Passing network framework that views different neural nets for graphs abstractly as message exchanges between nodes and message aggregation across nodes. They were defined for graph level classification tasks. Gilmer *et al.* (2017)'s framework for message passing is too abstract for any detailed analysis for node level tasks. Hence, we proposed a unified perspective of existing models as binomial graph kernels. This unified view gave us a new perspective of GCN's as binomial kernels over the current understanding of GCNs as stacked one layer Chebyshev Nets. It led us to derive a simple solution from improving the representation capacity of differentiable WL-based kernels. Moreover, it further aided in understanding why CONCAT (fusion) models are more powerful than the popular normalized attention mechanism.

4.5 Experiment results

We run detailed experiment to compare the proposed F-GCN model against GCNs with skip connections and GraphSAGE models with CONCAT combination. We follow the same experiment setup, evaluation metrics and implementation details as used with experiments for HOPF as in Section: 3.8. We evaluate F-GCN on the same set of datasets as HOPF but except for Reddit dataset. For the same hyperparameter setting, F-GCN's model was larger to be held in a 12GB GPU and henc we don't report for the same. In the following experimental analysis, F-GCN comes out as the consistent winner across 9/10 datasets losing out on only one dataset by an absolute 0.12 points

[1]Note that the work of Xu *et al.* (2018) appeared in parallel to ours in Arxiv, precisely our work appeared in Arxiv 11 days before theirs.

from the best.

Table 4.2: Transductive Experiments

	Datasets										Aggregate measures	
Models	Blog	FB	Movie	Cora	Citeseer	Cora2	Pubmed	Yeast	Human	Amazon	Shortfall	Rank
BL_NODE	37.929	64.683	50.329	59.852	65.196	40.583	83.682	59.681	41.111	64.121	9.431	5
GCN	39.101	63.682	51.194	77.523	71.903	63.152	86.432	60.34	62.057	73.746	1.235	2.8
GS-MEAN	39.433	62.651	50.557	76.821	70.967	62.8	84.23	59.771	63.753	68.266	2.223	3.8
GS-MAX	**40.275**	64.571	50.569	73.272	71.39	53.476	85.087	62.727	65.068	70.302	2.474	2.9
GS-LSTM	37.744	64.619	41.261	65.73	63.788	38.617	82.577	58.353	64.231	68.024	7.653	5.2
F-GCN	39.069	**64.857**	**52.021**	**79.039**	**72.266**	**63.993**	**86.5432**	**62.8479**	**65.538**	**74.097**	**0.121**	**1.3**

Table 4.3: Inductive learning experiment with Human dataset:

	NODE	GCN	GS-MEAN	GS-MAX	GS-LSTM	F-GCN
PPI	44.51	88.585	79.634	78.054	87.111	**88.942**

F-GCN Vs. NODE: F-GCN significantly outperforms the NODE model across the board.

F-GCN Vs. GCN: F-GCN improves over GCN in the inductive setup and outperforms GCN on nine of the ten datasets in the transductive setup while being comparable to the other one. F-GCN is seem to improve over GCN by upto 3.5 percentage points in Human dataset. On an average across all datasets, F-GCN provides 1.1 percentage points improvement.

F-GCN Vs GraphSAGE: F-GCN outperforms GraphSAGE variants across the board except for one where it slightly under-performs. GraphSAGE models have higher flexibility compared to GCNs as they have no shared weights. This explains the significant experimental improvement benefited with concatenation as noted in Hamilton *et al.* (2017). F-GCN significantly improves over GraphSAGE models in Human and Amazon dataset. There is no single winner among GraphSAGE models across datasets. Different variants champion in different datasets among them. *Despite having a single simple aggregation function, F-GCN easily champions over all of them combined except on BLOG.* This suggests that the flexibility to independently regulate information is necessary irrespective of the complex aggregation functions used. The mild lack in representation capacity holds back GraphSAGE from achieving F-GCN's performance.

Statistical Significance: We also evaluated the statistical significance of the overall results and the F-GCN model across multiple datasets with Friedman's and Wilcoxon sigend rank test (Demšar, 2006). With Friedman's test, we reject the null hypothesis

that all models are similar with p<0.05 on the transductive setup. F-GCN clearly beats the NODE only model and GS-LSTM on all datasets. With Wilcoxon signed-rank test, F-GCN $>$ GCN with $p < 0.01$, F-GCN $>$ GS-Mean with $p < 0.01$, F-GCN $>$ GS-Max with $p < 0.02$ and F-GCN $>$ GS-Max with $p < 0.01$

Overall, F-GCN improves over the state-of-the-art results on nine datasets while being extremely competitive on the other one. Though the proposed fusion component, in theory, is an optimal solution for GCNs with linear activations, they also seem to be experimentally beneficial for GCNs with the piece-wise linear ReLU activations too. Such generalization is not unrealistic in practice, as it is often observed that such generalization of insights from a relaxed linear analysis seems to provide significant clarity and potential improvements on the non-linear front (Saxe *et al.*, 2013; Kawaguchi, 2016; Hardt and Ma, 2016; Orabona and Tommasi, 2017).

F-GCN robustly captures mutli-hop information In real life datasets, there exists a varying amount of information among the interactions between the node and its different distant hop neighbors. An ideal relational model should be able to efficiently capture relevant information while filtering out the increasing noise induced by the expanding neighborhood size with each hop. We demonstrate F-GCN's capability in Fig: 4.3 to robustly capture information from multiple hops on different datasets with a varied information pattern. We selected only those datasets that have high relevant relational information for the classification task. These were datasets that obtained significant improvement on results over the node only classifier with just the inclusion of the first hop information.

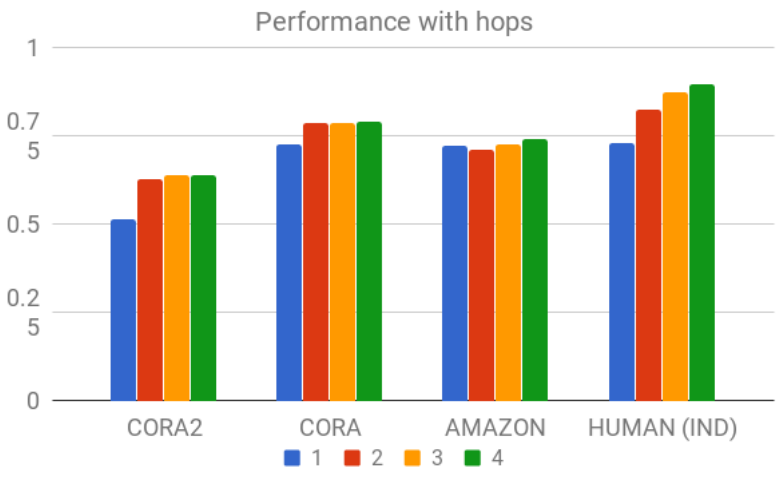


Figure 4.3: F-GCN performance with hops 1-4

In the citation networks (Cora and Cora2), it can be seen that the performances seem to saturate after two hops and with one hop in the Amazon, co-purchase network. Despite that, F-GCN with its capability to selectively regulate information from multiple hops remains unaffected by the noise induced by considering additional hops.

In contrast, there is a significant increase in performance with the consideration of nodes' higher order neighborhood interactions for the inductive experiment on the protein-protein interaction dataset (HUMAN). In the Human dataset, F-GCN was able to extract relevant information and achieve remarkable performance gain from further hops despite the dataset's high average degree.

F-GCN Vs NIP-MEAN F-GCN has a direct compuation path from h_0 to the label layer and hence it doesn't succumb to NIM issue. Table: 4.4 provides a results comparison among them. F-GCN beats NIP-MEAN on 8/10 datasets with NIP-MEAN outperforming F-GCN by more than 3% points on Human dataset. F-GCN has additional power to better capture multi-hop neighborhood information more effectively than NIP-MEAN model and thus results in providing more consistent results.

Earlier in 3.8, we noted that NIP kernel's ability to learn the correlation between the node and every other neighborhood feature might be important for the Human dataset. This is verified when we added Fusion component to the NIP-MEAN kernel, the model further improved NIP-MEAN's performance by 2.127% absolute points to achieve 70.815. However, on other datasets the results were not significantly different.

Table 4.4: FGCN Vs NIP kernel

Models	Blog	FB	Movie	Cora	Citeseer	Cora2	Pubmed	Yeast	Human	Amazon
NIP-MEAN	**39.433**	64.286	51.316	76.932	71.148	63.901	86.203	61.583	**68.688**	69.136
F-GCN	39.069	**64.857**	**52.021**	**79.039**	**72.266**	**63.993**	**86.5432**	**62.8479**	65.538	**74.097**

F-GCN benefits from Iterative learning The performance of F-GCN model can be further boosted by incorporating our proposed iterative learning component from the section: 3.4. We call this model as I-F-GCN. The performance benefits of iterative learning are provided in Table: 4.5 on datasets, where we observed significant improvements with Iterative models as discussed in the section: 3.9.4. I-F-GCN obtains an absolute improvement of 1.28-3.11% points over F-GCN and the CONCAT model proposed in our parallel work, Xu *et al.* (2018).

Table 4.5: I-F-GCN Vs FGCN

	Cora2	Yeast	Amazon
F-GCN	63.993	62.8479	74.097
I-F-GCN	**65.27**	**65.65**	**77.2086**

Comparison of Different Iterative Fusion models Though I-F-GCN provided considerable improvements over F-GCN, there seems to be no single winner among I-F-GCN and I-NIP-MEAN when compared with no additional hyper-parameter tuning. The results are available in Table: 4.6. I-FGCN without any additional Hyper-parameter tuning performs similar to I-NIP-MEAN on Cora2 and Amazon. However, there is a 4% drop in performance on the Yeast dataset. This can be attributed to the difference in choice of base graph kernel, i.e. GCN over NIP-MEAN. To understand the performance differences induced by this choice of graph kernel, we also evaluated I-F-NIP-MEAN, an Iterative Fusion model with NIP-MEAN as the base graph kernel. Component wise results for I-F-NIP-MEAN is available in Table: 4.7.

Table 4.6: I-F-GCN Vs I-F-NIP-MEAN

	Cora2	Yeast	Amazon
I-NIP-MEAN	**66.23**	**69.917**	75.045
I-F-GCN	65.27	65.65	**77.209**
I-F-NIP-MEAN	65.802	68.556	72.804

Jointly, we can argue that NIP-MEAN based Iterative models have a significant edge on the Yeast dataset over GCN based fusion model, I-F-GCN. Note, that though the linear fusion component in I-F-GCN provides an optimal first-order polynomial combination of the basis, the shortfall in performance against I-NIP-MEAN shows that powerful higher-order polynomial combination of the basis is required. A multi-layer Fusion step can a possible solution for this issue.

Table 4.7: NIP-MEAN with Fusion and Iterative leanring

	Cora2	Yeast	Amazon
NIP-MEAN	63.901	61.583	69.136
F-NIP-MEAN	64.32	62.29	68.67
I-F-NIP-MEAN	**65.802**	**68.556**	**72.804**

CHAPTER 5

Conclusion and Future Works

We proposed HOPF, a novel framework for Collective Classification that combines differentiable graph kernels with an iterative stage. Deep learning models for relational learning tasks can now leverage HOPF to use complete information from larger neighborhoods without succumbing to over-parameterization and memory constraints. HOPF can be improved by reducing erroneous information propagation through the use of pseudo-label information. It is always not the case that with every iterative inference step, the label estimates improve. Two classical approaches to solve this would be to (i) committing only high confidence labels as proposed in cautious ICA (McDowell *et al.*, 2007) and (ii) Adding ghost edges to increase the supervised information flow to unlabeled nodes (Gallagher *et al.*, 2008). Adaptions of these two approaches to HOPF would be highly impactful.

Through HOPF's modular framework we were able to better comprehend and compare numerous differentiable graph kernels. The generic kernel led us to see the hitherto issue of Node Information morphing and also allowed us to design solutions to fix the same with Node Information Preserving Kernel. Similarly, HOPF' generic graph kernel can be of help in designing newer and better graph kernels. For example, all the fixed and hand-crafted component which are unexplored currently, can now be parameterized and learned with stochastic gradient descent along with the other parameters of GCNs. The node's importance (α), neighbor's importance (β) and edge weights ($F(A)$) can be learned in this manner.

HOPF can be further extended for unsupervised tasks by incorporating structural regularization with Laplacian smoothing on the embedding space. With Laplacian regularizers, HOPF can be used for joint training of both Collective Classification and unsupervised network re-constructions loss which leads to obtaining superior node features that captures both attribute distribution in the neighborhood and the node's local network structure. Additionally, with the joint training setup, we can also parameterize higher order Laplacian regularizers and learn them with too.

In F-GCN work, we showed that differentiable recursive graph kernels are higher order binomial combinations of node and neighborhood information. Through analysis, we pointed out that such powerful recursively computed binomial functions lack the representation capacity to capture multi-hop neighborhood information effectively. Besides highlighting this critical issue, we also proposed a minimalist fusion component that can alleviate this issue. The current fusion component though learned, is the same for all nodes. This call for a new attention mechanism that can adaptively gives importance to information from multiple hops specific to binomial graph kernels.

REFERENCES

1. **Belkin, M.**, **P. Niyogi**, and **V. Sindhwani** (2006). Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of machine learning research*, **7**(Nov), 2399–2434.

2. **Bhattacharya, I.** and **L. Getoor** (2007). Collective entity resolution in relational data. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, **1**(1), 5.

3. **Bruna, J.**, **W. Zaremba**, **A. Szlam**, and **Y. LeCun** (2013). Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*.

4. **Cantador, I.**, **P. Brusilovsky**, and **T. Kuflik**, 2nd workshop on information heterogeneity and fusion in recommender systems (hetrec 2011). *In Proceedings of the 5th ACM conference on Recommender systems*, RecSys 2011. ACM, New York, NY, USA, 2011.

5. **Cavallari, S.**, **V. W. Zheng**, **H. Cai**, **K. C.-C. Chang**, and **E. Cambria**, Learning community embedding with community detection and node embedding on graphs. *In Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 377–386. ACM, 2017.

6. **Chakrabarti, S.**, **B. Dom**, and **P. Indyk**, Enhanced hypertext categorization using hyperlinks. *In ACM SIGMOD Record*, volume 27, 307–318. ACM, 1998.

7. **Chapelle, O.**, **B. Scholkopf**, and **A. Zien** (2009). Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]. *IEEE Transactions on Neural Networks*, **20**(3), 542–542.

8. **Cho, K.**, **B. Van Merriënboer**, **C. Gulcehre**, **D. Bahdanau**, **F. Bougares**, **H. Schwenk**, and **Y. Bengio** (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

9. **Craven, M.**, **A. McCallum**, **D. PiPasquo**, **T. Mitchell**, and **D. Freitag** (1998). Learning to extract symbolic knowledge from the world wide web. Technical report, Carnegie-mellon univ pittsburgh pa school of computer Science.

10. **Defferrard, M.**, **X. Bresson**, and **P. Vandergheynst**, Convolutional neural networks on graphs with fast localized spectral filtering. *In Advances in Neural Information Processing Systems*, 3844–3852. 2016.

11. **Demšar, J.** (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, **7**(Jan), 1–30.

12. **Frasconi, P.**, **M. Gori**, and **A. Sperduti** (1998). A general framework for adaptive processing of data structures. *IEEE transactions on Neural Networks*, **9**(5), 768–786.

13. **Gallagher, B.** and **T. Eliassi-Rad** (2010). Leveraging label-independent features for classification in sparsely labeled networks: An empirical study. *Advances in Social Network Mining and Analysis*, 1–19.

14. **Gallagher, B.**, **H. Tong**, **T. Eliassi-Rad**, and **C. Faloutsos**, Using ghost edges for classification in sparsely labeled networks. *In Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, 256–264. ACM, 2008.

15. **Gilmer, J.**, **S. S. Schoenholz**, **P. F. Riley**, **O. Vinyals**, and **G. E. Dahl** (2017). Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*.

16. **Glorot, X.** and **Y. Bengio**, Understanding the difficulty of training deep feedforward neural networks. *In Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 249–256. 2010.

17. **Gori, M.**, **G. Monfardini**, and **F. Scarselli**, A new model for learning in graph domains. *In Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, volume 2, 729–734. IEEE, 2005.

18. **Hamilton, W. L.**, **Z. Ying**, and **J. Leskovec**, Inductive representation learning on large graphs. *In Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, 1025–1035. 2017. URL http://papers.nips.cc/paper/6703-inductive-representation-learning-on-large-graphs.

19. **Hardt, M.** and **T. Ma** (2016). Identity matters in deep learning. *arXiv preprint arXiv:1611.04231*.

20. **Hatzis, C.** and **D. Page**, 2001 KDD cup challenge Dataset. *In pages.cs.wisc.edu/ dpage/kddcup2001*. KDD, 2001.

21. **He, K.**, **X. Zhang**, **S. Ren**, and **J. Sun**, Deep residual learning for image recognition. *In Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778. 2016.

22. **Hochreiter, S.** and **J. Schmidhuber** (1997). Long short-term memory. *Neural computation*, **9**(8), 1735–1780.

23. **Kawaguchi, K.**, Deep learning without poor local minima. *In Advances in Neural Information Processing Systems*, 586–594. 2016.

24. **Kingma, D. P.** and **J. Ba** (2014). Adam: A method for stochastic optimization. *CoRR*, **abs/1412.6980**. URL http://arxiv.org/abs/1412.6980.

25. **Kipf, T. N.** and **M. Welling** (2016). Semi-supervised classification with graph convolutional networks. *CoRR*, **abs/1609.02907**. URL http://arxiv.org/abs/1609.02907.

26. **Lancichinetti, A.**, **S. Fortunato**, and **J. Kertész** (2009). Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics*, **11**(3), 033015.

27. **Leskovec, J.** and **R. Sosič** (2016). Snap: A general-purpose network analysis and graph-mining library. *ACM Trans. Intell. Syst. Technol.*, **8**(1), 1:1–1:20. ISSN 2157-6904. URL http://doi.acm.org/10.1145/2898361.

28. **Levy, O.** and **Y. Goldberg**, Neural word embedding as implicit matrix factorization. *In Advances in neural information processing systems*, 2177–2185. 2014.

29. **Li, Y.**, **D. Tarlow**, **M. Brockschmidt**, and **R. Zemel** (2015). Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*.

30. **Lu, Q.** and **L. Getoor**, Link-based classification. *In Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, 496–503. 2003.

31. **Macskassy, S. A.** and **F. Provost** (2003). A simple relational classifier. Technical report, NEW YORK UNIV NY STERN SCHOOL OF BUSINESS.

32. **Mccallum, A.**, CORA Research Paper Classification Dataset. *In people.cs.umass.edu/ mccallum/data.html*. KDD, 2001.

33. **McCallum, A. K.**, **K. Nigam**, **J. Rennie**, and **K. Seymore** (2000). Automating the construction of internet portals with machine learning. *Information Retrieval*, **3**(2), 127–163.

34. **McDaid, A. F.**, **D. Greene**, and **N. Hurley** (2011). Normalized mutual information to evaluate overlapping community finding algorithms. *arXiv preprint arXiv:1110.2515*.

35. **McDowell, L. K.** and **D. W. Aha**, Semi-supervised collective classification via hybrid label regularization. *In Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*. 2012. URL http://icml.cc/2012/papers/513.pdf.

36. **McDowell, L. K.**, **K. M. Gupta**, and **D. W. Aha**, Cautious inference in collective classification. *In AAAI*, volume 7, 596–601. 2007.

37. **Moore, J.** and **J. Neville**, Deep collective inference. *In AAAI*, 2364–2372. 2017.

38. **Morris, C.**, **K. Kersting**, and **P. Mutzel**, Glocalized weisfeiler-lehman graph kernels: Global-local feature maps of graphs. *In Data Mining (ICDM), 2017 IEEE International Conference on*, 327–336. IEEE, 2017.

39. **Namata, G.**, **B. London**, **L. Getoor**, **B. Huang**, and **U. EDU**, Query-driven active surveying for collective classification. *In 10th International Workshop on Mining and Learning with Graphs*. 2012.

40. **Neumann, M.**, **R. Garnett**, **C. Bauckhage**, and **K. Kersting** (2016). Propagation kernels: efficient graph kernels from propagated information. *Machine Learning*, **102**(2), 209–245.

41. **Neville, J.** and **D. Jensen**, Iterative classification in relational data. *In Proc. AAAI-2000 Workshop on Learning Statistical Models from Relational Data*, 13–20. 2000.

42. **Neville, J.** and **D. Jensen**, Collective classification with relational dependency networks. *In Proceedings of the Second International Workshop on Multi-Relational Data Mining*, 77–91. 2003.

43. **Orabona, F.** and **T. Tommasi**, Training deep networks without learning rates through coin betting. *In Advances in Neural Information Processing Systems*, 2160–2170. 2017.

44. **Perozzi, B.**, **R. Al-Rfou**, and **S. Skiena**, Deepwalk: Online learning of social representations. *In Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 701–710. ACM, 2014.

45. **Pfeiffer III, J. J.**, **J. Neville**, and **P. N. Bennett**, Overcoming relational learning biases to accurately predict preferences in large scale networks. *In Proceedings of the 24th International Conference on World Wide Web*, 853–863. International World Wide Web Conferences Steering Committee, 2015.

46. **Poultney, C.**, **S. Chopra**, **Y. L. Cun**, *et al.*, Efficient learning of sparse representations with an energy-based model. *In Advances in neural information processing systems*, 1137–1144. 2007.

47. **Qiu, J.**, **Y. Dong**, **H. Ma**, **J. Li**, **K. Wang**, and **J. Tang** (2017). Network embedding as matrix factorization: Unifyingdeepwalk, line, pte, and node2vec. *arXiv preprint arXiv:1710.02971*.

48. **Rozemberczki, B.**, **R. Davies**, **R. Sarkar**, and **C. Sutton** (2018). Gemsec: Graph embedding with self clustering. *arXiv preprint arXiv:1802.03997*.

49. **Saxe, A. M.**, **J. L. McClelland**, and **S. Ganguli** (2013). Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*.

50. **Scarselli, F.**, **M. Gori**, **A. C. Tsoi**, **M. Hagenbuchner**, and **G. Monfardini** (2009). The graph neural network model. *IEEE Transactions on Neural Networks*, **20**(1), 61–80.

51. **Sen P, M.**, **Galileo**, **L. Getoor**, **B. Galligher**, and **T. Eliassi-Rad** (2008). Collective classification in network data. *AI magazine*, **29**(3), 93.

52. **Shervashidze, N.**, **P. Schweitzer**, **E. J. v. Leeuwen**, **K. Mehlhorn**, and **K. M. Borgwardt** (2011). Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, **12**(Sep), 2539–2561.

53. **Smola, A. J.** and **R. Kondor**, Kernels and regularization on graphs. *In Learning theory and kernel machines*. Springer, 2003, 144–158.

54. **Stark, C.**, **B.-J. Breitkreutz**, **T. Reguly**, **L. Boucher**, **A. Breitkreutz**, and **M. Tyers** (2006). Biogrid: a general repository for interaction datasets. *Nucleic acids research*, **34**(suppl_1), D535–D539.

55. **Subbanna, N.**, **D. Precup**, and **T. Arbel**, Iterative multilevel mrf leveraging context and voxel information for brain tumour segmentation in mri. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 400–405. 2014.

56. **Sun, W.**, **A. Venkatraman**, **B. Boots**, and **J. A. Bagnell**, Learning to filter with predictive state inference machines. *In International Conference on Machine Learning*, 1197–1205. 2016.

57. **Tibshirani, R.**, **G. Walther**, and **T. Hastie** (2001). Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **63**(2), 411–423.

58. **Tong, H.**, **C. Faloutsos**, and **J.-Y. Pan**, Fast random walk with restart and its applications. *In Proceedings of the Sixth International Conference on Data Mining*, ICDM '06, 613–622. IEEE Computer Society, Washington, DC, USA, 2006. ISBN 0-7695-2701-9. URL https://doi.org/10.1109/ICDM.2006.70.

59. **Tu, C.**, **W. Zhang**, **Z. Liu**, and **M. Sun**, Max-margin deepwalk: Discriminative learning of network representation. *In IJCAI*, 3889–3895. 2016.

60. **Vaswani, A.**, **N. Shazeer**, **N. Parmar**, **J. Uszkoreit**, **L. Jones**, **A. N. Gomez**, **Ł. Kaiser**, and **I. Polosukhin**, Attention is all you need. *In Advances in Neural Information Processing Systems*, 5998–6008. 2017.

61. **Wang, S.**, **J. Tang**, **C. Aggarwal**, and **H. Liu**, Linked document embedding for classification. *In Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, 115–124. ACM, 2016.

62. **Wang, X.**, **P. Cui**, **J. Wang**, **J. Pei**, **W. Zhu**, and **S. Yang**, Community preserving network embedding. *In AAAI*, 203–209. 2017.

63. **Wang, X.**, **L. Tang**, **H. Gao**, and **H. Liu**, Discovering overlapping groups in social media. *In Data Mining (ICDM), 2010 IEEE 10th International Conference on*, 569–578. IEEE, 2010.

64. **Weisfeiler, B.** and **A. Lehman** (1968). A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, **2**(9), 12–16.

65. **Xu, K.**, **C. Li**, **Y. Tian**, **T. Sonobe**, **K.-i. Kawarabayashi**, and **S. Jegelka** (2018). Representation learning on graphs with jumping knowledge networks. *arXiv preprint arXiv:1806.03536*.

66. **Yang, C.**, **Z. Liu**, **D. Zhao**, **M. Sun**, and **E. Y. Chang**, Network representation learning with rich text information. *In IJCAI*, 2111–2117. 2015.

67. **Yang, Z.**, **W. W. Cohen**, and **R. Salakhutdinov** (2016*a*). Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861*.

68. **Yang, Z.**, **W. W. Cohen**, and **R. Salakhutdinov**, Revisiting semi-supervised learning with graph embeddings. *In Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, 40–48. 2016*b*. URL http://jmlr.org/proceedings/papers/v48/yanga16.html.

69. **Yu, Q.** and **D. A. Clausi**, Combining local and global features for image segmentation using iterative classification and region merging. *In Computer and Robot Vision, 2005. Proceedings. The 2nd Canadian Conference on*, 579–586. IEEE, 2005.

70. **Zhu, X.** and **Z. Ghahramani** (2002). Learning from labeled and unlabeled data with label propagation.

PUBLICATIONS

List of papers based on thesis

1. Priyesh Vijayan, Yash Chandak, Mitesh Khapra, Srinivasan Parthasarathy and Balaraman Ravindran; **HOPF: Higher Order Propagation Framework for collective classification**; *Eighth International workshop on Statistical Relational AI* co-located with IJCAI 2018, Stockholm, Sweden.

2. Priyesh Vijayan, Yash Chandak, Mitesh Khapra, Srinivasan Parthasarathy and Balaraman Ravindran; **Fusion Graph Convolutional Networks**; *Fourteenth International workshop on Mining and Learning with Graphs* co-located with KDD 2018, London, UK.

List of papers not based on thesis

1. Anasua Mitra*, Priyesh Vijayan*, Srinivasan Parthasarathy and Balaraman Ravindran; **Learning clusterable graph embeddings with Semi-Supervised NMF**; *First International workshop on Relational Representation learning* co-located with NIPS 2018, Montreal, Canada. (*equal contribution)

CURRICULUM VITAE

Full Name	Priyesh Vijayan
Date of Birth	12-05-1992
E-mail	priyesh@cse.iitm.ac.in
	priyeshvellore@gmail.com
Website	https://priyeshv.github.io
Education	M.S. by Research (CSE)
	Indian Institute of Technology, Madras
Permanent Address	Door no: 6
	Rajaji street, Bharathi Nagar Extn,
	Katapdi, Vellore
	Tamil Nadu - 632007.

GENERAL TEST COMMITTEE

Chairman Dr. P Sreenivasa Kumar
Department of Computer Science and Engineering
Indian Institute of Technology, Madras.

Guide Dr. Balaraman Ravindran
Department of Computer Science and Engineering
Indian Institute of Technology, Madras.

Members Dr. Mitesh Khapra
Department of Computer Science and Engineering
Indian Institute of Technology, Madras.

Dr. A.N. Rajagopalan
Department of Electrical Engineering
Indian Institute of Technology, Madras.