Exploring Local Information for Graph Representation Learning

The University Of Sheffield.

Li Zhang

Department of Computer Science

University of Sheffield

This dissertation is submitted for the degree of

Doctor of Philosophy

April 2022

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Li Zhang

April 2022

Acknowledgements

First and foremost, I would like to express my gratitude to my supervisor, Dr. Haiping Lu, for his consistent guidance, support, patience. I want to thank him for giving me the opportunity to do this PhD under his supervision.

I would like to thank Dr Nikolaos Aletras, Dr. Jun Ma and Yan Ge for their invaluable advice, guidance and close research collaboration. I sincerely thank my panel: Mark and Eleni for their invaluable suggestions and guidance of my PhD progress. I also thank the China Scholarship Council and University of Sheffield PhD program for providing financial supports.

Special thanks to my friends in the Machine Learning group who have been great colleagues: Shuo, Xianyuan, Chao, Lawrence, Johanna, Chunchao, Peizhen, Juan, Tianqi, Wenwen, Fariba, Wil.

Finally, I want to thank my love, my parents for their endless love and support.

Abstract

Graphs are important data structures that can capture interactions between individual entities. The primitive graph representation is usually high-dimensional, sparse, noisy, and in irregular forms, which is challenging for direct usage in downstream tasks (e.g., node or graph classification). Various graph representation learning (GRL) techniques have been developed to convert the raw graph data into low-dimensional vector representations while preserving the intrinsic graph properties. Graph neural networks (GNNs) are currently the most popular paradigm that can utilize nodes' local information to assist their representation learning.

Local neighborhood information in graphs varies greatly for different nodes. Therefore, direct neighborhood aggregation in GNNs is not an optimal choice. This thesis aims to develop new GNNs by exploring and modeling different types of local information to tackle the weaknesses of current GNNs in both algorithms and applications. We first propose three new models: 1) node feature convolution for graph convolutional network (NFC-GCN) to consider feature-level attention of local information; 2) learnable aggregator for GCN (LA-GCN) to generalize NFC-GCN further by lifting constraints on the input data format; and 3) hop-hop relation-aware GNN (HHR-GNN) to incorporate hop-level attention of local information. Moreover, we apply HHR-GNN to two industrial graphs for personalized video search and cross-domain recommendation tasks. Experimental studies show that the proposed methods have outperformed related state-of-the-art methods in both standard tasks of node/graph classification as well as application-specific tasks of search and recommendation.

Important Notations and Abbreviations

Symbol	Definition
$\mathbf{A}$	Adjacency matrix
$\mathbf{A}_r$	Adjacency matrix for r-th type edge
$\hat{\mathbf{A}}$	Normalized adjacency matrix
α_{ij}	Weight between node i and node j
C_{onv}	Convolution operation
d_i	Node degree of node i
$\mathbf{d}_d$	Document d' embedding
$\mathbf{d}_d^{'}$	Learned document d' embedding
$\mathbf{D}$	Diagonal degree matrix
$\mathcal{D}(x, y)$	Discriminator
$f_{ag}^{(k)}$	Aggregation function in the k-th layer of a model
$\mathcal{G}$= $(\mathcal{V}, \mathcal{E}, \mathbf{X})$	Graph, nodes, edges, node feature
h,r,t	Head entity, relation, tail entity
$\mathbf{h}$, $\mathbf{r}$, $\mathbf{t}$	Head, relation, tail embedding
$\mathbf{h}_i^{(k)_r}$	$r-$hop representation
$\mathbf{h}_i^{(l)}$	Hidden representation of node i
$\mathbf{H}^{(l)}$	Hidden output of $l-th$ layer
$\mathbf{I}_N$	Identity matrix
$g_{\mathbf{w}}$	A function of eigenvalues

$\mathcal{I}_{\mathcal{S}}$	Source domain
$\mathcal{I}_{\mathcal{T}}$	Target domain
k	Filter size
$\mathbf{L}$	Laplacian matrix
Λ	Diagnose matrix
$\lambda_0, ..., \lambda_{N-1}$	Eigenvalues of laplacian matrix
$\mathcal{N}_i$	Neighborhood of node i
p	The number of hops
$\mathbf{q}_q$	Query q' embedding
$\mathbf{q}_q^{'}$	Learned query q' embedding
$r_{us}^{\mathcal{S}}$	Predict score for source domain
$r_{ut}^{\mathcal{T}}$	Predict score for target domain
s	Stride
sim_{ij}	Cosine similarity between node i and node j
$\mathbf{u}_u$	User u' embedding
$\mathbf{u}_u^{'}$	Learned user u' embedding
$\mathbf{W}^{(l)}$	Weight matrix in the $l-th$ layer
$\mathbf{X}_i^{'}$	Local feature map
$\hat{y}_c$	Click score
$\hat{y}_r$	Correlation score between query and document
$\mathbf{Y}_{lf}$	Label indicator matrix

Abbreviation	Description
CDR	Cross-Domain Recommendation
CF	Collaborative Filtering
CG	Click Graph
CMF	Collective matrix factorization
CNN	Convolutional Neural Network
GAT	Graph Attention Network
GCN	Graph Convolutional Network
GNN	Graph Neural Network
GNNs-He	GNNs for heterogeneous graphs
GNNs-Ho	GNNs for homogeneous graphs
GRL	Graph Representation Learning
HeG	Heterogeneous Graph
HoG	Homogeneous Graph
HHR-GNN	Hop-Hop Relation-aware Graph Neural Network
IR	Information Retrieval
KG	Knowledge Graph
KG-NeuCMF	KG-aware Neural Collec-tive Matrix Factorization
LA-GCN	Learnable Aggregator for GCN
MI	Mutual Information
MF	Matrix Factorization
MLP	Multilayer Perceptron
NCF	Neural Collaborative Filtering
NA	Neighborhood Aggregation
NFC	Node-Feature Convolution
NLP	Natural Language Processing
PVS	Personalized Video Search
RNN	Recurrent Neural Network
RS	Recommender Systems

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Graphs, such as social networks, knowledge graphs and citation networks, are ubiquitous data structures that can capture interactions (edges) between individual entities (nodes). Except the structure (connectivity or topology) information, nodes in a graph are typically associated with feature vectors [73, 240]. For example, in a citation network, nodes represent documents, edges represent citations between documents, and node features represent textual information, often as bag-of-words, i.e., sparse vectors of weighted word frequencies in a document. Graphs can be broadly divided into homogeneous graphs (HoG, one type of nodes and edges) and heterogeneous graphs (HeG, multiple types of nodes and edges) [218, 233].

Effective graph analysis can help us to understand various and complex graphs in a systematic manner [113]. For example, by analysing graphs, we can recommend new friends to a user in a social network and classify the role of a protein in the biological interaction graph [72]. There are two general approaches for inference on graph datasets [154]. The first is statistical relational learning (SRL) that develops statistical methods to model graph datasets. Some representative methods include Probabilistic Relational Models (PRM), Relational Dependency Networks(RDNs) Bayesian Logic Programming (BLP),Markov logic networks (MLN)[64]. Generally, these methods model the dependency of objects (nodes) using conditional random fields [108], however the inference of these models are quite challenging due to the complicated relational structures among objects (nodes). Another line research is based on the graph representation learning (GRL) methods, the key is to

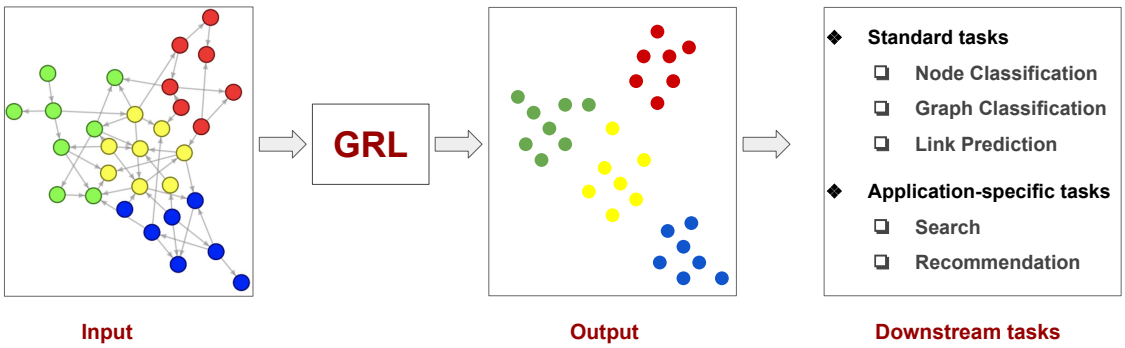


Fig. 1.1 The general GRL framework. The key idea of GRL is to encode the original high-dimensional and sparse representation of each node (input) into a low-dimensional space, which can preserve the information of the original graph as much as possible. Then the learned embeddings can be used in downstream tasks such as node/graph classification and so on.

learn graph representations that map nodes, or entire (sub)graphs to another low-dimensional space, which can reflect the information of the original graph as much as possible. Graph representation learning (GRL) can automatically learn to encode graph structures into low-dimensional embeddings, which has been widely recognized as an effective approach to graph analysis. The learned representation can be used in downstream machine learning systems and aid in standard tasks such as node classification [103], graph classification [215], link prediction [161], or application-specific tasks such as search [179, 169], recommendation [77, 106]. Figure 1.1 shows the GRL framework that takes a high-dimensional and sparse graph as an input in downstream tasks.

GRL methods can be categorized into four groups: matrix factorization based approaches [157, 16, 5, 26, 145], random walk based approaches [151, 152], neural network based approaches and graph neural networks (GNNs) approaches [103, 73]. The former three types of methods are limited to transductive settings and cannot incorporate node features in the learning process [225]. Different from previous methods, GNNs are currently the most popular paradigm, largely owing to their efficiency, inductive learning capability and their ability to combine graph structure and node features simultaneously in the learning process [72, 191].

The GNN framework mainly consists of two key steps: neighborhood aggregation and feature transformation, where nodes aggregate and transform the local neighborhood information in each layer [72, 215]. By stacking multiple GNN layers, information can be propagated further through the graph structure and we can embed nodes into low-dimensional representations, which naturally combines graph structure and node features in the learning process. This schema iteratively updates the representation of a node by aggregating representations from its local neighborhood and transformation, which can also be treated as a general neural message-passing process [65]. The obtained representation can be fed into down-stream tasks. GNNs have been successfully applied in various tasks, such as node classification [103, 191, 72], link prediction [238], recommender system [189, 201, 205], knowledge graph completion [161, 9].

1.1 Motivation and Research Questions

The key of GNNs is to utilize a node's local neighborhood information to assist its representation learning. Local neighborhood information in graphs varies greatly for different nodes. Therefore, direct neighborhood aggregation in GNNs is not an optimal choice. This thesis aims to develop new GNN algorithms that can learn better node or graph representations by exploring and modeling different types of local information. Before describing research questions, I will give a brief background about them and detailed one will be presented in Chapter 2.

- Current GNN models are designed with the homophily assumption that connected nodes are likely to share the same label. However, real-world graphs are noisy and adjacent nodes do not necessarily imply similarity. Some strategies have been proposed that allow for node-level attention of local neighborhood information, however all individual features in a node feature vector are still treated equally. Each feature within a neighbor feature vector may play a different role for the central node's representation learning. Therefore, the first research question (**Q1**) is *how to design*

the GNN architecture that can consider both node-level and feature-level attention of the adjacent neighbors in the learning process?

- For node representation learning, only the first-order neighborhood information may be not enough, which is caused by the sparsity or heterophily. Instead of only utilizing adjacent neighbors, GNNs can also leverage multi-scale neighborhood information to assist a given node's representation learning. Intuitively, neighbors from different hops show different importance for the central node's representation learning. However, existing methods usually aggregate the central node with different hops of neighbors directly, the real meaningful relations between central node and neighbors from different hops have not been explored by current models. Therefore, the second research question (**Q2**) is *how to design the GNN architecture that can incorporate hop-level attention of local neighborhood information in the learning process?*

- Industrial graphs, such as click graphs (CGs), knowledge graphs (KGs), are usually large-scale, sparse, and heterogeneous (containing different types of nodes and edges), which are quite challenging to deal with. The goal of GRL is to encode the intrinsic structural and semantic properties of the input graph into the low-dimensional latent embedding vectors, thus can benefit application-specific tasks (personalized video search and cross-domain recommendation). Therefore, the third research question (**Q3**) is *how to apply the proposed GNN for application-specific tasks by leveraging the local neighborhood with rich semantic information?*

1.2 Structure and Contributions

This section outlines the structure of this thesis, alongside presenting the key novel contributions as shown in Fig. 1.2. The first chapter introduces the overview of GNNs based graph representation learning and our motivation. Chapter 2 presents a survey of the literature relating to the topics explored in this thesis. After that, this thesis is organized into two parts largely. Part I contains three chapters that present three new GNN algorithms and Part II

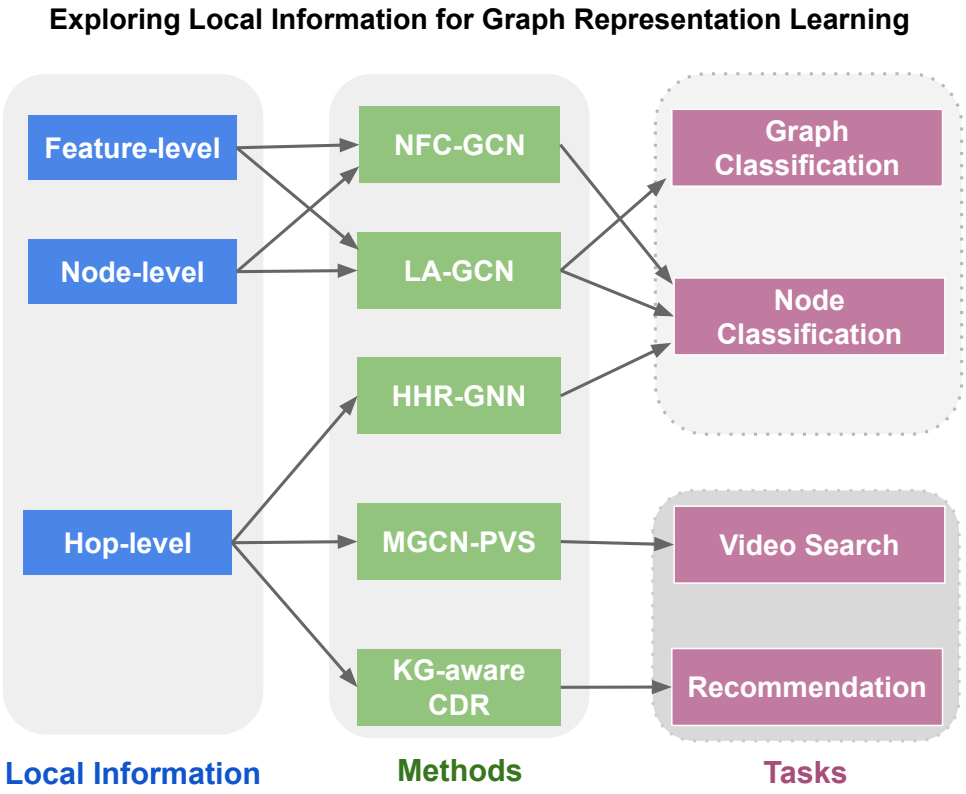


Fig. 1.2 The structured contributions of this thesis is that exploring different local information for graph representation learning, which is applied to both standard tasks (node/graph classification) and application-specific tasks (search and recommendation).

contains two chapters that present how to apply the proposed GNN to industrial graphs (click graphs, knowledge graphs) and applications (personalized video search and cross-domain recommendation). Finally, Chapter 8 summarises the findings and contributions of this thesis. This chapter also presents future research directions in the area based on the work presented in the prior chapters. The major contents of Chapters 2-7 are summarized as follows.

Chapter 2: Background presents a survey of the literature relating to the topics explored in this thesis. This chapter begins with describing definitions of homogeneous graphs, heterogeneous graphs, graph representation learning, and graph neural networks. It then presents representative GNN extensions for both homogeneous and heterogeneous graphs, which will help to describe the motivation of works in Part I. Additionally, this chapter reviews related technical skills: convolution operation and knowledge graph embedding that

will be used in Part I. Finally, two industry tasks: personalized video search and cross-domain recommendation are introduced, which will help to describe the motivations of works in Part II.

In Part I: Algorithms, we introduce three new GNN models on node and graph classification, which are important and standard graph mining tasks.

- **Chapter 3: Node-Feature Convolution for Graph Convolutional Networks** presents a new node-feature convolutional (NFC) layer for GCN. The NFC layer first constructs a feature map using features selected and ordered from a fixed number of neighbors. It then performs a convolution operation on this feature map to learn the node representation. In this way, we can learn the usefulness of both individual nodes and individual features from a fixed-size local neighborhood. This chapter also introduces experiments on node classification task to show the superior performance of our algorithms over existing methods.

- **Chapter 4: A Feature-Importance-Aware and Robust Aggregator for GCN** extends the NFC-GCN model to a more flexible and general framework by lifting constraints on the input data format. Specially, we unify current learnable aggregators in a general framework: Learnable Aggregator for GCN (LA-GCN). Under this framework, this chapter presents a new model called LA-GCN$_{\text{Mask}}$ that learns a specific mask for each neighbor of a given node, allowing both node-level and feature-level attention of local neighborhood information. Additionally, experiments on seven graphs for node classification and graph classification tasks in this chapter show the superior performance of our algorithm over existing models.

Contribution 1: We propose two new GNN models: NFC-GCN and LA-GCN$_{\text{Mask}}$ to solve Q1. NFC-GCN performs a convolution operation on the fix-sized feature map constructed by the central node and selected neighbors, and LA-GCN learns a specific mask for each neighbor of a given node, which allows the two models can consider both node-level and feature-level attention of local neighborhood information in the learning process.

Chapter 5: Hop-Hop Relation-aware Graph Neural Networks proposes to leverage knowledge graph embedding (KGE) methods to automatically learn the relationship (relation-score) between a given node and its different hops of neighbors, which allows for hop-level attention of local neighborhood information. This is a more flexible and general local neighborhood aggregation framework that can be applied to both homogeneous and heterogeneous graphs representation learning. Finally, experimental results in this chapter for node classification on both types of graphs show the competitive performance on both accuracy and efficiency of our model compared to state-of-the-art GNNs models.

Contribution 2: We propose a new aggregation framework: HHR-GNN to solve Q2. HHR-GNN leverages KGE models to learn the relation-scores between a given node and its different hops of neighbors, and this mechanism can achieve the hop-level attention of local neighborhood information in the learning process.

In Part II: Applications, we apply and modify HHR-GNN proposed in Chapter 5 for two real-world applications: personalized video search and cross-domain recommendation.

- **Chapter 6: A GCN-based Multi-task Learning Framework for Personalized Video Search** proposes to apply GNNs to enhance the user, query and document representation learning by utilizing their local neighboring nodes from the user-query graph and click graph. We apply the hop-aware projection strategy as in HHR-GNN to capture a given node's first-order (i.e., user-query) and second-order (i.e., user-user)

interactions in the graphs. Considering the real-industry graphs with millions of nodes and edges, this chapter also presents an efficient, localized convolution by sampling fixed-size neighbors from graphs. Finally, we conduct extensive experiments on a large-scale real dataset obtained from a well-known video search platform. Experimental results show that our proposed model can significantly outperform most state-of-the-art personalized search methods.

- **Chapter 7: Knowledge-aware Cross-Domain Recommendation** proposes to apply knowledge graph (KG) to assist the cross-domain recommendation. To this end, we first construct a new dataset *AmazonKG4CDR* from the Freebase and a subset of Amazon in three domains: movies, books, and music. Then, this chapter presents a new framework, KG-aware Neural Collective Matrix Factorization (KG-NeuCMF) by leveraging KG to enrich item representations. It first applies the relation-specific projection as in HHR-GNN to learn item embeddings to capture both domain-specific and domain-general knowledge from adjacent and higher-order neighbors in the KG. To further improve KG-aware item embeddings, we maximize the mutual information between representations learned from the KG and user-item matrix. Finally, we conduct extensive experiments on the newly constructed dataset and demonstrate that our model significantly outperforms the best-performing baselines.

  Contribution 3: It is the first time to apply GNNs to two industry tasks: personalized video search (PVS) and cross-domain recommendation (CDR), where the user relationship, click graphs for PVS and knowledge graphs for CDR can be utilized to assist the two tasks. Considering the industry graphs are heterogeneous and contain millions of nodes and edges, we apply the sampling strategy and hop-aware (relation-specific) projection to efficiently and effectively leverage the local neighborhood information to assist the central node's representation learning (Q3).

Chapter 8: Conclusion and Future work summarises our findings and contributions of this thesis. We also present future research directions in the area based on the work presented

in the prior chapters. The contents of this thesis are based on the following publications and papers that are currently under review, for which I am a leading author. The technical contents of Chapters 4 has appeared in ACM copyright materials, with the permission to preprint granted by ACM.

1. Li Zhang, Heda Song, Nikolaos Aletras, and Haiping Lu. "Node-Feature Convolution for Graph Convolutional Networks." Pattern Recognition (2022).

2. Li Zhang, and Haiping Lu. "A Feature-Importance-Aware and Robust Aggregator for GCN", in Proceedings of the 29th ACM Conference on Information and Knowledge Management (CIKM 2020), Galway, Ireland, pp. 1813-1822, (2020).

3. Li Zhang, Yan Ge, and Haiping Lu. "Hop-Hop Relation-aware Graph Neural Networks.".

4. Li Zhang, Lei Shi, Jiashu Zhao, Juan Yang, Tianshu Lyv, Dawei Yin and Haiping Lu. "A GCN-based Multi-task Learning Framework for Personalized Video Search". in Proceedings of the 15th ACM International Web Search Data Mining Conference, 2022.

5. Li Zhang, Ge Yan, Jun Ma, Jianshu Zhao, Jianmo Ni, Lei Shi, Dawei Yin and Haiping Lu. "Knowledge-aware Neural Collective Matrix Factorization for Cross-domain Recommendation." .

Chapter 2

Background

In Chapter 1, we have shown the importance of graph representation learning and the most effective approaches: GNNs. Therefore, in this chapter, we start with definitions of different types of graphs and classical GRL models. After that, we introduce graph neural network and representative GNN extensions for both homogeneous and heterogeneous graphs to gain insights of the shortcomings of current GNNs. Additionally, this chapter reviews related technical skills: convolution operation and knowledge graph embedding that will be used in Part I. Finally, two application-specific tasks: personalized video search and cross-domain recommendation are introduced for Part II.

2.1 Graphs

Definition 2.1.1. ***Graph***. A graph with N nodes can be represented as $\mathcal{G}$= $(\mathcal{V}, \mathcal{E}, \mathbf{X})$, where node $v_i \in \mathcal{V}$, edges $(v_i, v_j) \in \mathcal{E}$ $(i, j = 1, ..., N)$, and a feature matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$ containing N D-dimensional feature vectors. An adjacency matrix $\mathbf{A}$ is commonly used to represent the structure of a graph and matrix entries indicate the relationships among nodes.

Graphs, such as social networks, knowledge graphs and citation networks, are ubiquitous data structures that can capture interactions between individual nodes as defined in Definition 2.1.1. In a citation network, nodes represent documents, edges represent citations between documents, and node features represent textual information often as bag-of-words, i.e., sparse

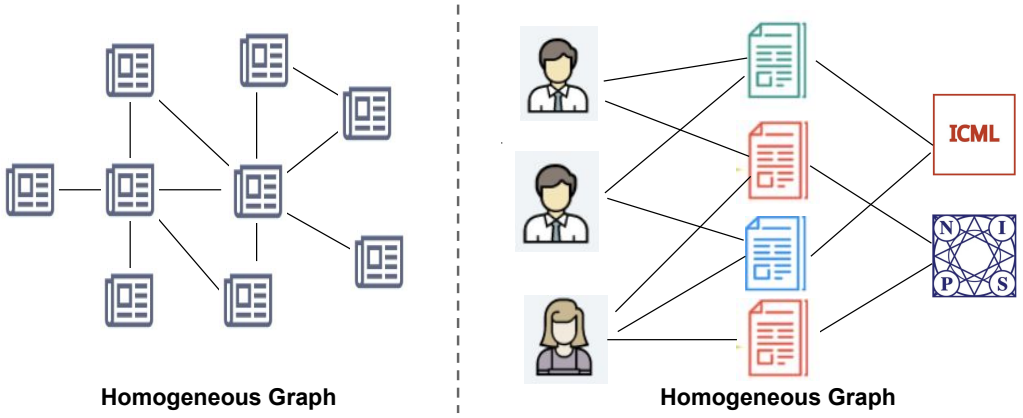


Fig. 2.1 Homogeneous Graph and Heterogeneous Graph.

vectors of weighted word frequencies in a document. Graphs can be broadly divided into homogeneous graphs (HoG) and heterogeneous graphs (HeG) [218, 233] defined as follows:

Definition 2.1.2. ***Homogeneous Graph*** [218]. Homogeneous graph is a graph with one types of nodes and links.

Definition 2.1.3. ***Heterogeneous Graph*** [218]. Heterogeneous graph is a graph with multiple types of nodes and links, each node is associated with a node type, and each link is associated with a link type. It is worth noting that the type of a link $\mathcal{E}_{ij}$ automatically defines the types of nodes v_i and v_j on its two ends.

As shown in Fig .2.1, in the homogeneous graph (citation graph), there is only one type of node (Paper) and edge (Citation), and there exists three types of nodes (Author, Paper, Conference) and two types of edges (AP, PC) in the heterogeneous graph.

Heterogeneous graph is associated with a node type mapping function f_v: $\mathcal{V} \rightarrow \mathcal{T}_v$ and a link type mapping function f_e: $\mathcal{E} \rightarrow \mathcal{T}_e$, where $|\mathcal{T}_v| + |\mathcal{T}_e| > 1$. If both $\mathcal{T}_v$ =1 and $\mathcal{T}_e$ =1, it is a homogeneous graph with the same type of nodes and edges. The heterogeneous graph can be represented by a set of adjacency matrices $\{\mathbf{A}_r\}_{r=1}^{R}$ (R=$|\mathcal{T}_e|$), and $\mathbf{A}_r \in \mathbb{R}^{N\times N}$ is an adjacency matrix where $\mathbf{A}_r[i,j]$ is non-zero when there is a r-th type edge from v_j to v_i. In homogeneous graphs, the adjacency matrix is simplified to $\mathbf{A} \in \mathbb{R}^{N\times N}$ (R=1).

Nodes or edges affiliated with various attributes are commonly observed in graphs besides the topological structure, such as the node features and edge types information [40, 57, 134].

In this view, both the homogeneous graphs (with node features) and the heterogeneous graphs can be both called attributed graphs.

2.2 Graph Representation Learning

Definition 2.2.1. ***GRL*** [218]. Given a graph denoted as $\mathcal{G}$= $(\mathcal{V}, \mathcal{E}, \mathbf{X})$, GRL is a mapping function $f : v_i \rightarrow \mathbf{h}_i \in \mathbb{R}^d$,where $d \ll |\mathcal{V}|$. The objective function is to make the similarity between $\mathbf{h}_i$ and $\mathbf{h}_j$ explicitly preserve the connectivity information of v_i and v_j.

Graph representation learning (GRL) aims to represent each node in a graph or a graph into low dimensional vector(s), meanwhile preserve the graph structure and the inherent properties of the graph, as defined in Definition 2.2.1. The learned embeddings can be used in various down-stream tasks such as node classification, link prediction, clustering [103, 161, 223]. Fig. 1.1 shows a general GRL pipeline that takes high-dimensional and sparse connectivity information as input to learn the low-dimensional and dense vectors, which can be used in different downstream tasks.

Graph representation leaning methods can be categorized into the factorization-based, random walk-based and neural network-based approaches [73].

2.2.1 Matrix Factorization-based GRL

Early methods for learning representation for nodes mainly focus on matrix factorization (MF) approaches. They are directly inspired by classic techniques for dimensionality reduction [16, 5, 157, 98].

In adjacency matrix, a row vector or column vector can be used as the vector representation of a node, but the formed representation space is N-dimensional, where N is the total number of nodes. Matrix factorization-based GRL aims to learn a low-dimensional vector space for each node, in contrast with the N-dimensional space, meanwhile preserves the pairwise similarity. For v_i and v_j, the pairwise node similarity of $\mathbf{h}_i$ and $\mathbf{h}_j$ should

approximate the original pairwise similarity and the loss function can be defined as:

$$\mathcal{L} = \sum_{(v_i,v_j)\in\mathcal{V}} \|\mathbf{h}_i^T \mathbf{h}_j - d(v_i,v_j)\|^2. \tag{2.1}$$

$d(v_i,v_j)$ is the original similarity of v_i and v_j. There are a large number of GRL methods, such as Laplacian Eigenmaps (LE) [16], Graph Factorization (GF), GraRep [5], and HOPE [145] all fall firmly within this class and they differ primarily in the used pairwise similarity measure $d(v_i,v_j)$. GF defines $d(v_i,v_j)$ based on the adjacency matrix $\mathbf{A}_{ij}$. GrapRep is based on the powers of the adjacency matrix, and HOPE is based one neighborhood overlaps.

2.2.2 Random Walk-based GRL

The adjacency matrix $\mathbf{A}$ contains the topology of graph and every node's representation, however each node representation is usually a sparse, discrete, and high-dimensional vector due to the nature of sparseness in large-scale networks, which is very similar with the word representation in the natural language processing (NLP) [151]. The development of Word2Vector [135, 136], significantly improves the effectiveness of word representation by transforming sparse, discrete and high-dimensional vectors into dense, continuous and low-dimensional vectors.

Inspired by Word2Vec, DeepWalk [151] generates random paths over a graph. It regards a node as a word, a random path as a sentence, and the neighborhood as the context. Word2Vec applies the SkipGram model that maximizes the co-occurrence probability among the words that appear within a window size in a sentence [135]. Given the representation $\mathbf{h}_i$ of v_i, DeepWalk aims to maximize the average logarithmic probability of its neighbors in the walk, It can be described as following:

$$\mathcal{L} = \sum_{(v_i\in\mathcal{V})} \sum_{v_i'\in\mathcal{W}_{v_i}} -log(P(\mathbf{h}_i' \mid \mathbf{h}_i)). \tag{2.2}$$

$\mathcal{W}_{v_i}$ is a uniformly random walk taking v_i as the root. $P(\mathbf{h}_i' \mid \mathbf{h}_i)$ can be calculated as following:

$$P(\mathbf{h}_i' \mid \mathbf{h}_i) = \frac{exp(\mathbf{h}_i^T \mathbf{h}_i')}{\sum_{v_k \in V} exp(\mathbf{h}_i^T \mathbf{h}_k)}. \tag{2.3}$$

The success of DeepWalk motives many subsequent studies [153]. DeepWalk relies on simple unbiased random walks over the graph, while Node2vec [70] adopts a more flexible walk strategy by introducing two hyper-parameters to determine depth versus breadth of the walk. LINE explicitly models the first-order and second-order graph proximity instead of a fixed-length random walks [181]. Besides, [153] learns node embeddings with truncated random walks that *skip* some nodes in a window size. [34] measures the similarity by using a hyperbolic rather than Euclidean, distance measure.

2.2.3 Neural Network-based GRL

Given the huge success of deep learning techniques, efforts have been made to generalize them to graphs, which have opened a new chapter of graph representation learning. In [116], gated recurrent units are introduced in the propagation step. The neural graph fingerprints method [50] further introduces a convolution-like propagation rule. PATCHY-SAN [143] selects and normalizes a fixed-size neighborhood for a given node, then CNNs can be applied to learn the neighborhood structure information. Structure Deep Network Embedding (SDNE) [198] extends the traditional deep autoencoder [193] to learn from the node structure information to get a low-dimensional embedding for each node. Besides these mentioned methods, graph neural networks (GNNs) are currently the most popular methods and we will introduce GNNs in Sec. 2.3 specially.

2.3 Graph Neural Networks

The matrix factorization-based, random walk-based and neural network-based methods (such as PATCHY-SAN, SDNE) only utilize graph structure (without node features or attributes) to learn new node representations. Graph neural networks (GNNs) have been first introduced in [68]. They consist of an iterative process which propagates the node states until the

node representation reaches a stable fixed point. In recent years, convolutional neural networks (CNNs) [109] have been successfully applied to tackle problems such as image classification[107], [99] semantic segmentation [66] or machine translation [63]. Inspired by this, there is an increasing interest in generalizing CNNs to the graph domain. These convolutional networks on graphs are now commonly known as Graph Convolutional Neural Networks (GCNNs) or simply GNNs.

2.3.1 Convolution on Graphs

The convolution operation in CNN is only defined for regular grids, thus it requires special designs of localized convolutional filters on graphs due to the irregular connectivity of nodes in a graph. Convolution on graphs fall into two categories, spectral-based and spatial-based. Spectral-based approaches define graph convolutions by introducing filters from the perspective of graph signal processing [168] where the graph convolutional operation is interpreted as removing noises from graph signals. Spatial-based approaches inherit ideas from information propagation. In this section, we will introduce the spacial convolution, spectral convolution on graphs, then the representative GNN extensions for both homogeneous and heterogeneous graph representation learning.

Spacial Convolution. Analogous to the convolutional operation of a conventional CNN on an image, spatial-based methods define graph convolutions based on a node's spatial relations [50, 143, 138]. One of the challenges of these approaches is to define an operator which works with different sized neighborhoods and maintains the weight sharing property of CNNs. In [50], the authors used five different weights at each hidden layer for nodes with different node degree. The local message-passing architecture works well for the low degree of organic molecules, but it can not be applied to large graphs with wide node degree distributions. [11] extends CNN to graphs by introducing "diffusion-convolution" operation, and it defines the neighborhood by using the power of a transition matrix and learns the weights for each input channel and neighborhood degree. [143] constructs a fixed neighborhood, which can be treated as the receptive field for a node, then the CNN

architecture can be used. [138] presents mixture model CNNs (MoNet), a spatial approach which provided a unified generalization of CNN architectures to graphs.

Spectral Convolution. The spectral convolutions on graphs [25] can be defined as the multiplication of a signal $\mathbf{x} \in \mathbb{R}^N$ with a filter $g_{\mathbf{w}}$ parameterized by $\mathbf{w} \in \mathbb{R}^N$ in the fourier domain as following:

$$g_{\mathbf{w}} \star \mathbf{x} = \mathbf{U} g_{\mathbf{w}}(\Lambda) \mathbf{U}^T \mathbf{x}. \tag{2.4}$$

$g_{\mathbf{w}}$ can be seen as a function of eigenvalues of the graph laplacian matrix $\mathbf{L}$, and $\mathbf{L}$ is defined as

$$\mathbf{L} = \mathbf{I}_N - \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}} = \mathbf{U} \Lambda \mathbf{U}^T, \tag{2.5}$$

where $\mathbf{I}_N$ is an identity matrix, $\mathbf{D}$ is a diagonal degree matrix with $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$. $\mathbf{U}$ is the matrix of eigenvectors of the $\mathbf{L}$. $\Lambda = diag([\lambda_0, ..., \lambda_{N-1}]) \in \mathbb{R}^{N \times N}$ and $\lambda_0, ..., \lambda_{N-1}$ are eigenvalues of $\mathbf{L}$. Eq. (2.4) incurs expensive computation of the laplacian eigenvectors ($\mathcal{O}(N^2)$).

To circumvent this problem, Chebyshev Spectral CNN (ChebNet) [42] applies a polynomial function to approximate $g_{\mathbf{w}}(\Lambda)$. The Chebyshef polynomial is recursively defined as $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$, with $T_0(x) = 1$ and $T_1(x) = x$. Then a filter can be parameterized as the truncated expansion with order K as following:

$$g_{\mathbf{w}}(\Lambda) \approx \sum_{k=0}^{K} \mathbf{w}_k T_k(\tilde{\Lambda}), \tag{2.6}$$

where $\tilde{\Lambda} = 2\Lambda/\lambda_{max} - \mathbf{I}_N$ is a diagonal matrix of scaled eigenvalues in [-1,1]. The filter operation can be written as:

$$g_{\mathbf{w}} \star \mathbf{x} \approx \sum_{k=0}^{K} \mathbf{w}_k T_k(\tilde{\mathbf{L}}) \mathbf{x}, \tag{2.7}$$

where $\tilde{\mathbf{L}} = 2\mathbf{L}/\lambda_{max} - \mathbf{I}_N$. $T_k(\tilde{\mathbf{L}})$ can be calculated using the recurrence relation: $T_k(\tilde{\mathbf{L}}) = 2\tilde{\mathbf{L}} T_{k-1}(\tilde{\mathbf{L}}) - T_{k-2}(\tilde{\mathbf{L}})$, and the entire filter operation cost is $\mathcal{O}(KN)$.

GCN [103] further simplifies Eq. (2.7) and limits $K = 1$, and the filter parameter $\mathbf{w}_k$ is shared over the whole graph. GCN bridges the gap between spectral-based approaches and spatial-based approaches, which has developed rapidly recently due to its attractive efficiency,

flexibility, and generality. For signal $\mathbf{X} \in \mathbb{R}^{N \times D}$ with D input channels (i.e., a D-dimensional feature vector for every node) and D_1 filters, Eq. (2.7) (the convolved signal matrix) can be written as:

$$\mathbf{H}^{(1)} = \sigma\left(\hat{\mathbf{A}}\mathbf{X}\mathbf{W}^{(0)}\right), \tag{2.8}$$

where $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$ is a normalized adjacency matrix of the undirected graph $\mathcal{G}$ with added self-connections $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I_N}$, $\tilde{\mathbf{D}}$ is defined with its diagonal entries as $\tilde{\mathbf{D}}_{ii} = \sum_j \tilde{\mathbf{A}}_{ij}$, $\mathbf{W}^{(0)} \in \mathbb{R}^{D \times D_1}$ is a trainable input-to-hidden weight matrix, $\sigma(\cdot)$ denotes an activation function, such as the $\mathrm{ReLU}(\cdot) = \max(0, \cdot)$, and $\mathbf{H}^{(1)} \in \mathbb{R}^{N \times D_1}$ is the matrix of activation in the first layer. Thus, the propagation rule can be written as:

$$\mathbf{H}^{(l+1)} = \sigma\left(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right), \tag{2.9}$$

where $\mathbf{W}^{(l)} \in \mathbb{R}^{D_{l-1} \times D_l}$ is a layer-specific trainable weight matrix and $\mathbf{H}^{(l)} \in \mathbb{R}^{N \times D_l}$ is the matrix of activation in the l-th layer. $\mathbf{H}^{(0)} = \mathbf{X}$ is the node feature matrix.

In Eq. (2.9), a GCN layer can be divided into two steps: (1) aggregating the given node and its neighbors' feature vectors with different weights (according to the node degrees):

$$\hat{\mathbf{h}}_i^{(l)} = \mathbf{h}_i^{(l)} + \sum_{j \in \mathcal{N}_i} \frac{1}{\sqrt{d_i d_j}} \mathbf{h}_j^{(l)} (j \in \mathcal{N}_i), \tag{2.10}$$

where d_i and d_j are the node degrees of node v_i and node v_j respectively, and $\mathbf{h}_i^{(l)}$, $\mathbf{h}_j^{(l)}$ are the representations of v_i, v_j of the l-th layer, $\mathcal{N}_i$ is the neighborhood of v_i in the graph; (2) feeding the averaged feature vector to a fully-connected neural network.

2.3.2 GNNs for Homogeneous Graph Representation Learning

The majority of current GNNs focus on homogeneous graphs (GNNs-HO), and we review two types of GNN extensions that are closed related to our model in Chapter 3, 4 and 5.

Sampling-based methods. Eq. 2.10 shows that GCN learns a new node representation from features of all its neighbors. In real-world graphs, the number of neighbors for a given node can range from one to hundreds or even thousands. Therefore, some nodes may not have

sufficient number of neighbors to aggregate information, while some other nodes may have their own features $\mathbf{h}_i^{(l)}$ being "washed out" due to aggregating information from too many neighbors [115]. Moreover, varying number of neighbors can lead to neighborhood explosion which subsequently causes computational problems, e.g., excessive space (memory) [103] and time complexity [72, 36].

Instead of considering all neighbors, some methods apply sampling strategies to only aggregate a part of neighbors [123]. GraphSAGE [72] uniformly samples a fixed number of neighbors and aggregates them with a sum, mean, LSTM or maxpooling aggregator as follows:

$$\hat{\mathbf{h}}_i^{(l)} \approx \mathbf{h}_i^{(l)} + aggregator(\mathbf{h}_j^{(l)}, j \in \hat{\mathcal{N}}_i), \tag{2.11}$$

where $\hat{\mathcal{N}}_i$ is the neighborhood generated by a fixed-length random walk, and they can come from a different number of hops, or search depth, away from a given node. FastGCN [36] interpretes graph convolutions as integral transforms of embedding functions and directly sampled the nodes in each layer independently. JP-networks [216] samples learned intermediate representations for a given node to get the final node representation. LLC [209] proposes to learn layer-wise connections instead of using all the intermediate representations as in JP-networks. PASS [224] proposes to choose the best sampling policy, e.g., the dot product similarity measure, or concatenation-based measure, or random choice, by propagating gradients through the non-differentiable sampling operation. [37] provides the a theoretical analysis for sampling-based models and also proposes a generic and efficient doubly variance reduction schema.

Besides sampling strategies, some dropout tricks are proposed. DropEdge [156] randomly removes a set of edges. Graph DropConnect (GDC) [75], and GeniePath [124],learns the connections in a graph, jointly with GNN model parameters. PTDNet [127] prunes the graph edges by penalizing the number of task-irrelevant edges in the graph with parameterized networks. [148] shows analyze the theoretical properties of dropout-based GNN in details.

These sampling-based methods mainly focus on how to select neighbors, but do not treat the selected neighbors differently in the latter aggregation step.

Neighbor weighting-based methods. GCN aggregates neighbors with fixed weights inversely proportional to the central node and neighbors' node degrees. Once the graph structure is given, the weights are fixed. Many algorithms have been proposed to treat the neighbors differently in neighborhood aggregation process. Disentangled graph convolutional network (DisenGCN) [129] proposes a neighborhood routing mechanism to identify the factor that may have caused the link from a given node to one of its neighbors, and accordingly send the neighbor to the channel responsible for that factor. Then each channel can perform an aggregation independently, which means each *cluster* of neighbors are treated differently in DisenGCN. Inspired by attention mechanisms [190], Graph Attention Networks (GAT) introduces an attention mechanism to dynamically assign weights to different neighbors [191] as:

$$\hat{\mathbf{h}}_i^{(l)} = \mathbf{h}_i^{(l)} + \sum_{j \in \mathcal{N}_i} \alpha_{ij} \mathbf{h}_j^{(l)}, \tag{2.12}$$

where α_{ij} is the learned weights with a shared attention mechanism. GATv2 [24]further extends GAT to a more expressive dynamic attention mechanism and structure learning graph attention Networks (SLGAT) [231] attentions both the directly linked and higher-order neighbors.

Although each node is treated differently, all individual features in a feature vector share the same weight, without considering their *individual importance*. Masked GCN [219] learns a diagonal mask matrix that can determine which attributes can be propagated to the central node, LA-GCN [235] and GNN-Film respectively introduce an auxiliary model and "feature-wise linear modulations" (FiLM) [150] for a feature-wise modulation in the neighborhood aggregation process. The two methods considered all neighbors, which is not necessary and directly learning the mask or FiLM need a very huge model, especially for graphs with high-dimensional node features. Learnable Graph Convolutional Layer (LGCL) [58] applies CNN on the *reorganized embeddings* (learned from GCN) of the central node and its neighbors, rather than the original features. The reorganization of node embeddings breaks the original correspondence between node representations.

2.3.3 GNNs for Heterogeneous Graph Representation Learning

Learning models on heterogeneous graphs requires different considerations to effectively represent their node and edge heterogeneity. Recently, some works have attempted to model the heterogeneous graph by using GNNs [55, 86, 47, 218]. RGCN [161] and HetGNN [233] use either distinct linear projection weight or type-specific RNN to encode features for each type of adjacent neighbors. Based on RGCN, R-HGNN [226] adds a cross-relation message passing module to improve the interactions of node representations across different relations. Some methods [244, 118] propose to learn the heterogeneous graph structure and model parameter simultaneously. However, these models do not consider central nodes' high-order neighbors. Another type of algorithms utilize meta-paths to model higher-order proximity and the mata-path is defined as:

Definition 2.3.1. Meta-path [178]. A meta-path is a path defined on the network schema in a form of $v_1 \xrightarrow{r_1} v_2 \xrightarrow{r_2} \ldots \xrightarrow{r_p} v_{p+1}$, where v and r are node types and link types, respectively.

Definition 2.3.1 defines a composite relation $R = r_1 \circ r_2 \ldots \circ r_p$ between node v_1 and v_{p+1}. Given the composite relation R, the adjacency matrix of the meta-path can be obtained by multiplications of adjacency matrices as:

$$\mathbf{A}_R = \mathbf{A}_{r_1}\mathbf{A}_{r_2}\ldots\mathbf{A}_{r_p}. \tag{2.13}$$

For example, two Authors can be connected by the meta-path Author-Paper-Author (APA) via the meta path $A \xrightarrow{AP} P \xrightarrow{PA} A$ (a two-step walk), and the co-author (AA) graph can be obtained by the multiplication of $\mathbf{A}_{AP}$ and $\mathbf{A}_{PA}$.

Given the manually defined meta-paths, Heterogeneous Graph Attention Network (HAN) [206] only aggregates two end nodes along the meta-path, which results in information loss. Moreover, this is a two-stage approach and requires hand-crafted meta-paths, which will significantly affect the performance of downstream tasks. Some models propose to learn the meta-path, Graph Transformer Networks (GTN) [164] learns a soft selection of edge types for generating multiple meta-paths, RMS-HRec [144] trains a reinforcement learning (RL) based policy network to identify the high-quality meta-paths, but these learned meta-paths are still

general rules and may not be suitable for every node in a graph. Besides, the intrinsic design and implementation (e.g., sampling neighbors based on the manually defined meta-paths, or recalculate the new adjacency matrix in each training epoch) in GNNs-HE make them very slow when modeling large-scale heterogeneous graphs.

2.3.4 Comparison of Different GRL Methods

In this subsection, we compare the advantages and disadvantages of the mentioned GRL methods.

MF-based models view the graph representation learning as a low-dimensional approximation of the pairwise (node-node) similarity, these models usually assume that connected nodes should learn similar embeddings in the embedding space. Many classical dimensionality reduction methods can be applied, which are mathematically transparent. However, they failed to capture higher-order relationships among nodes. The key difference between random walk-based and MF-based models is how they define the notion of node similarity. Random walk-based methods propose more flexible ways to define the node similarity and two nodes have similar embeddings if they tend to co-occur on short random walks over the graph, which can simultaneously preserve both local and higher-order node similarities.

MF-based and random walk-based methods have achieved many successes in the past decade, however these models suffer from some drawbacks. (1) No parameters are shared between nodes in the learning process. The learned model can be seen as an an embedding lookup table based on the node ids, and it means the model's parameters grow with the number of nodes in a graph, which is computationally inefficient. (2) These models are inherently transductive and can not deal with unseen nodes, unless additional rounds of optimization are performed to optimize the embeddings for these nodes. This is highly problematic to generalize to new graphs after training. (3) These methods fail to leverage the node feature information in the learning process, which is often highly informative with respect to the node's structure information in the graph, such as the textual information of the papers in the paper citation graph.

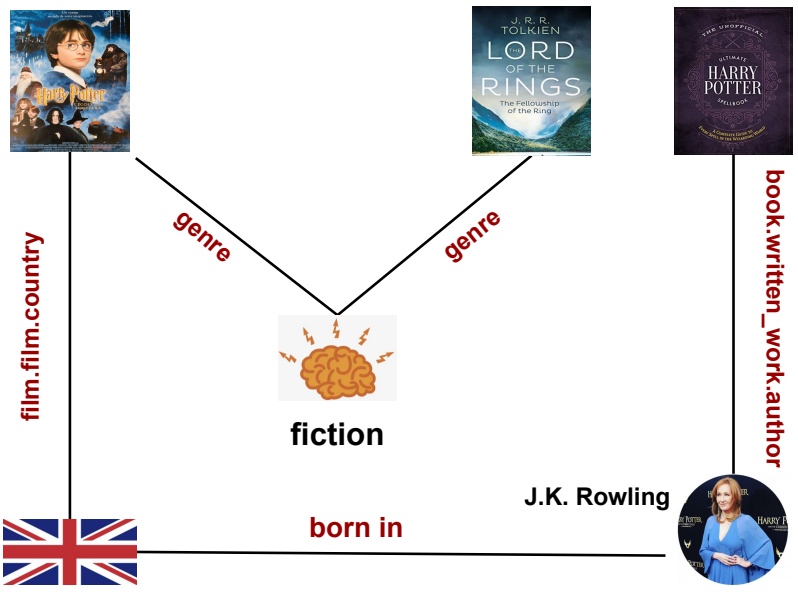


Fig. 2.2 An example of a knowledge graph that contains entities and relations.

In the past years, many research works have applied deep learning approaches for graph data, and GNNs are currently the most popular paradigm. Compared with the MF-based, random-walk based models, GNNs are more efficiency, due to the parameter sharing mechanism among nodes. Different with MF-based, random-walk based models, GNNs are inductive algorithms and can leverages node feature information (e.g., text attributes) to efficiently generate node embeddings for previously unseen data. Finally, GNNs can combine graph structure and node features simultaneously in the learning process.

The key of GNNs is to utilize a node's local neighborhood information to assist its representation learning. Local neighborhood information in graphs varies greatly for different nodes and how to leverage the local neighborhood information to assist both classical graph-related tasks and application-specific tasks have rarely been exploited by current GNNs. Thus, this thesis aims to develop new GNN algorithms that can learn better node or graph representations by exploring and modeling different types of local information.

2.4 Knowledge Graph Embedding

Knowledge graphs (KGs) contain rich knowledge in the form of heterogeneous graphs where nodes correspond entities and edges correspond to relations. Recent years have witnessed rapid growth in KG construction and application. A large number of KGs, such as Freebase [18], DBpedia [112], YAGO [175], and NELL [28], have been created and successfully applied to many real-world applications, from recommendation system (RS) [200, 205], semantic parsing [80], to information extraction [80] and question answering [19, 22]. Knowledge in KGs is presented as in the form of the triple (*head entity, relation, tail entity*), also called a fact [204]. For example, in Fig. 2.2, (*J.K. Rowling, born in, UK*) indicates that J.K. Rowling was born in the UK.

KGE methods aim to model the relationship between *head entity* and *tail entity* in knowledge graph (KG) and assign a score of how likely it is that two entities are in a certain relationship [171, 204]. KGE models can be roughly divided into three groups: translational distance models [21, 208, 121, 93]) and semantic matching models (RESCAL-based [142, 217, 141, 187] and neural network-based methods [171, 44, 20]).

Translational models are popular to exploit distance-based energy functions and a relation is regarded as a translation in the embedding space. TransE [21] is the most representative translational distance model. Given a fact (*h,r,t*), the embedded entities $\mathbf{h}$ and $\mathbf{t}$ can be connected by $\mathbf{r}$ with low error and the scoring function is defined as the distance between $\mathbf{h}+\mathbf{r}$ and $\mathbf{t}$:

$$f(h,t) = ||\mathbf{e}+\mathbf{r}-\mathbf{t}||. \tag{2.14}$$

However, TransE has flaws in dealing with 1-to-N, N-to-1, and N-to-N relations. To overcome the disadvantages of TransE, TransH [121] and TransR [93] introduce the relation-specific hyperplane and relation-specific spaces respectively. The entitie embeddings $\mathbf{h}$ and $\mathbf{t}$ are first projected to the hyperplane or the space specific to relation r, and the scoring function follows the general idea of TransE.

Semantic matching models exploit similarity-based scoring functions. They measure plausibility of facts by matching latent semantics of entities and relations embodied in their

vector space representations [204]. In RESCAL [142], each relation is represented as a matrix which models pairwise interactions between latent factors. The score of a fact (*h,r,t*) is defined

$$f(h,t) = \mathbf{h}^T \mathbf{M}_r \mathbf{t}, \tag{2.15}$$

where **h** and **t** are representations of the head and tail, and $\mathbf{M}_r$ is a matrix associated with the relation. DistMult [217] is actually simplified from RESCAL by restricting $\mathbf{M}_r$ to diagonal matrices.

Neural network-based models conduct semantic-matching using neural network architectures. MLP is a simpler approach and the $head$, $tail$, $relation$ vectors (**h**, **r**, **t**) are concatenated and fed to the MLP and mapped to a non-linear hidden layers. The score is then generated by a linear output layer shown as follows:

$$f(h,t) = \mathbf{W}^{(T)} MLP(||(\mathbf{h},\mathbf{r},\mathbf{t})). \tag{2.16}$$

where $\mathbf{W}^{(T)}$ is the learnable weight matrix in the linear output layer, $||$ denotes concatenation. Neural Tensor Network (NTN) [171] is more expressive that can relate two input entities vectors across multiple dimensions and each slice of the tensor is responsible for one type of entity pair. In the recent years, new emerging research areas are focusing on applying GNN based models to the entity embeddings in the knowledge graph [199, 207, 48, 252, 39]. RGCN [161] uses nodes' neighborhood information for learning entity representations. [165, 117] leverages relation information to compute different weights to the neighboring nodes for learning embeddings of entities and relations. However, the GNN-based methods are limited to the transductive setting, GraIL [185] proposes an inductive way that can generalize to unseen entities after training. Considering most of current methods are designed for static graphs, some papers extent GNN-based models to dynamic knowledge graphs [213, Xu et al.].

2.5 Personalized Video Search

2.5.1 Video Search

Traditional solutions for video search are based on matching the query with the video title [228]. The title-based matching is a text-to-text matching problem, which has been well solved through existing information retrieval algorithms. Nevertheless, the title-based search relies on high-quality video titles, which are not always available in real applications. To alleviate the issues caused by low-quality video titles, an alternative solution is to understand the visual content of the video. With the success on both visual and textual representation learning, many cross-model methods have been proposed to calculate the relevance between a textual search query and a visual content, which is much more challenging than the intra-modal text-to-text search task [51]. They can be coarsely divided into two groups based on their architectures: one-stream model [176, 6] that text and video are together feed into an encoder and two-stream model [248, 59, 56, 228] that one encoder encodes the text information and another encoder encodes the video (or segments of the video) information. Then the relevance between the query and the video is determined by the similarity between their global features [8, 67, 27, 90, 246].

Different from Web search, video search is mainly for entertainment and users' preferences could be very diverse because of their different backgrounds. For example, when issuing the query "Welcome to New York", a music fan may want to find the music video from Taylor Swift, while a film fan may want to watch the film directed by Abel Ferrara. Similar to many other vertical search areas, there is a strong need to return personalized search results to different users for the same query in video search.

2.5.2 Personalized Search

Search engine has become a common tool for people to obtain information from the web. Due to differences in individual preferences, the same query often represents different query intents. For example, when issuing the query *Welcome to New York*, a music fan may want to find the music video from *Taylor Swift*, while a film fan may want to watch the film directed

by *Abel Ferrara*. There is a strong need to return personalized search results to different users for the same query. Existing personalized search methods (PSMs) achieve personalization by building user profiles with their meta information or search history to refine the original ranking.

The main idea of traditional static personalized search algorithms in [49, 183, 184] is that they evaluate the click probability by counting the number of documents clicked by the same user under the same query. Some studies attempt to extract the topics features by utilizing open directory project (ODP) [169, 210] or latent semantic analysis(LSA) or latent dirichlet allocation (LDA) techniques [29, 74, 194, 195]. But these topic-based models are often trained in an unsupervised manner, thus the performance of these models on personalized search is not as good as expected. Recently, deep learning methods have been successfully applied to a variety of language and information retrieval applications. One category of DL-based PSMs mainly follow the general framework of traditional DL search methods [89], meanwhile incorporate more personal information. [88] encodes both textual information of the query, and users' social connections [221] or location [87] to represent the query. Another category methods aim to learn the user profile from users' long-term or short-term search history by bi-LSTM, RNN, GRU or transformer to learn users' search intents [4, 60, 126, 249].

One of the biggest concerns in personalized video search is the data sparsity issue, where each user only has a handful of queries in their search histories and therefore limits the learning capability of the personalization algorithms. Besides, current PSMs learn the query and document representation only from their textual content. When queries are short and vague, it is difficult for current PSMs to learn accurate representations. What's more, current PSMs only use the click signal to train their models and this manner is not suitable in our scenario, because the real industry video platform exists much noisy click information.

2.5.3 Graphs based Information Retrieval

Graph-based information has been widely studied and exploited in information retrieval (IR) [146]. Click graph is a bipartite graph extracted from click-through data. It contains

two types of nodes: queries and documents and edges in the graph means click for query-document pairs by any user [14]. Click graph can enrich the current query and document and provide more search context to help disambiguation, which has been successfully applied in query-to-document retrieval, query/document clustering and classification [96, 38, 3]. For personalized search, users perform as the most important role. Click graph only reveal the connections among queries and documents. The relationship among users should not be ignored, which can be extremely helpful to overcome the sparsity problem of users' history. Different users can issue the same queries, so the user-query graph can be extracted from click-through data. Incorporating both query-document graph and user-query graph in the learning process can greatly benefit the personalized video search task.

2.6 Knowledge Graph for Recommendation

2.6.1 Knowledge Graph for Recommendation

Recommender systems (RS) have become a popular technique in many applications, e.g., Youtube (video sharing), Amazon (e-commerce), and Facebook (social networking), as they provide suggestions of items to users so that they can avoid facing the information overload problem [155]. In recent years, introducing recommendations with the KG as side information has attracted considerable interest [200, 204, 205]. A KG is a heterogeneous graph, where nodes represent as entities, edges represent relations between entities and a fact in KG is usually represented in the form of a triple (*head entity*, *relation*, *tail entity*) [204]. KGs contain rich semantic relatedness among items and incorporating KGs in RS can help explore the latent connections and provide explanations for recommended items [71]. Recent years, KG-aware RS models are mainly for the single-domain RS [31, 200, 182, 245, 204, 205]. SemStim [78] proposes to apply knowledge graph to the cross-domain recommendation. It exploits the semantic links found in a knowledge graph (e.g. DBpedia), to find indirect paths between the source and target domains. Then, these found products in the target domains will be returned as the recommendations, ranked by their activation levels.

2.6.2 Cross-Domain Recommendation

Current RS models, such as collaborative filtering (CF) [160], suffer from the sparsity issue as the real-world datasets usually have a long tail of users and items with few feedbacks. To address the data sparsity problem, cross-domain recommendation (CDR) [54] been proven to be a promising method to alleviate the sparsity.

Different from conventional single-domain recommendation, CDR can leverage information from source domain to improve the performance of target domain [17, 54], namely single-target CDR, which is a powerful tool to deal with the data sparsity problem. These approaches extend the single-domain recommendation models by utilizing same contents, such as tags, reviews [53, 229], common items or users [170, 84, 119] as the bridge and transferring information between domains [85, 125, 158, 159]. The single-target CDR approaches only focus on how to leverage the source domain to help improve the recommendation accuracy on the target one, but not vice versa. Recently, dual-target CDR mothods [131, 250, 114] has been proposed to improve the performance on both source and target domains simultaneously by leveraging dual-transfer learning strategies [241, 76]. However, as referred in Negative Transfer [147], this idea does not work well, because, in principle, the knowledge learned from the sparser domain is less accurate than that learned from the richer domain, and thus the recommendation accuracy on the richer domain is more likely to decline by simply and directly changing the transfer direction. Therefore, dual target CDR models demand novel and effective solutions. In summary, none of current CDR models can indeed improve the performance on both domains simultaneously, and they are significantly hindered by limited information and connections between two domains.

Part I

Algorithms

Chapter 3

Node-Feature Convolution for Graph Convolutional Network

3.1 Introduction

As described in Sec. 2.3.1, the GCN architecture represents a node by aggregating the feature vectors of all its neighbors, analogous to the receptive field of a *convolutional kernel* in Convolutional Neural Networks (CNNs) [110]. However, the number of neighbors typically varies (e.g., from one to hundreds) across nodes. Directly aggregating all the adjacent neighbors as in GCN (without considering the number of neighbors) may lead to neighborhood explosion which subsequently causes computational problems, e.g., excessive space (memory) [103] and time complexity [72, 36]. Besides, the importance of individual neighbors of a node is fixed (depending on the node degree), lacking the flexibility to characterize different relationships between nodes. Finally, individual features in a neighbor feature vector may have different usefulness, while they share the same weight in GCN. This may result important features can not obtain enough attention, and vice versa, which is apparently not an optimal aggregation strategy. Figure 3.1 shows an example. The central node 0 belongs to Class A (*Neural Networks*) and it can be cited (i.e., connected) by papers from Class B (*Probabilistic Methods*). Node 5 from Class B may contain some common features with the central node 0 from Class A, e.g., *neuron*, and also some features more

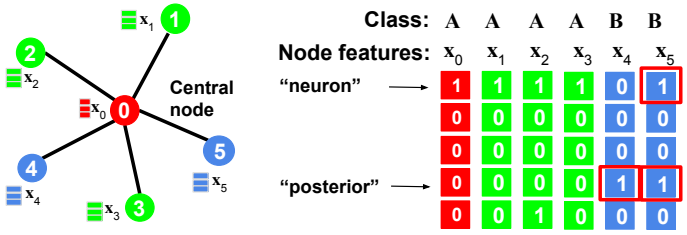


Fig. 3.1 A six-node subgraph from the Cora dataset [132]. Each node corresponds to a machine learning paper, with a bag-of-words feature vector $\mathbf{x}_i$ ($i = 0, 1, 2, ..., 5$). Nodes 0–3 belong to Class A (*Neural Networks*), and nodes 4–5 belong to Class B (*Probabilistic Methods*). Individual features in $\mathbf{x}_i$ are not equally important for representing the central node 0.

unique for Class B, e.g., *posterior*. Thus, the feature *neuron* should be more important than *posterior* for representing the central node. However, these features are equally weighted in existing GCN methods.

Some extensions, as introduced in Sec. 2.3.3, have been proposed to address the mentioned limitations: (1) sampling-based methods sample a fixed-size set of neighbors or learn an adaptive receptive fields for the given node; (2) neighbor weighting-based methods learn to treat different neighbors differently instead of simple aggregation. However, the sampling-based methods can not treat different features within a feature vector differently and directly weighting all neighbors may bring in too much noisy information and further influence the result. Besides, weighting each neighbors will be time consuming and unnecessary, especially for dense graphs. In summary, existing works do not solve all the limitations together.

In this thesis, we propose a novel method called Node-Feature Convolution for Graph Convolutional Network (NFC-GCN) [237] to solve all the mentioned problems. Our method learns to assign different weights to individual node features to get a new representation of a given node in three steps and we summarise our contributions as follows:

- We apply the neighbor selection strategy to select the most important neighbors for neighborhood aggregation, which alleviates the neighborhood explosion problem caused by aggregating too many neighbors.

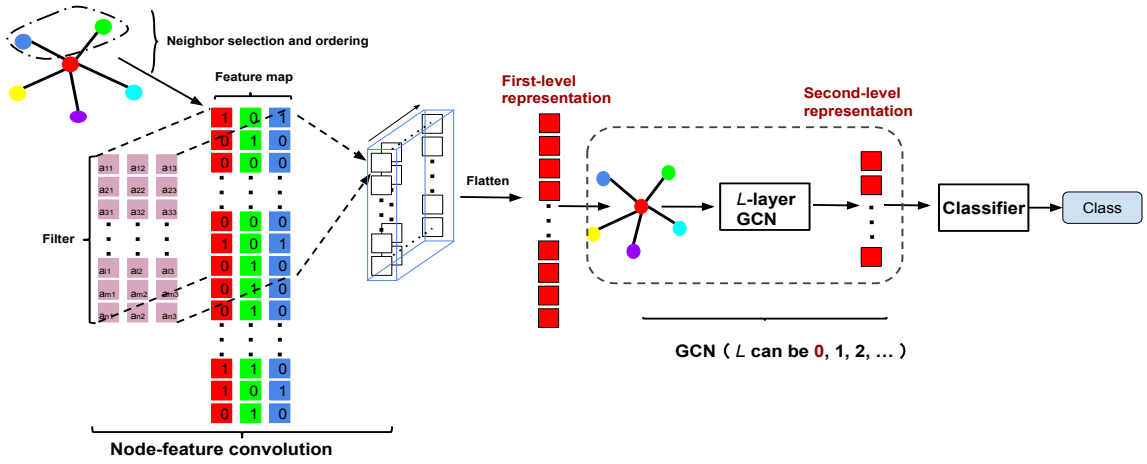


Fig. 3.2 NFC-GCN architecture. NFC-GCN consists of three main steps: (1) *Neighbor selection and ordering*; (2) *Node-feature convolution* operating on node-feature maps to obtain a flattened first-level node representation; and (3) *GCN:* the first-level NFC representation is passed through an L-layer GCN model (L is a hyperparameter) to learn a second-level node representation is passed to a classifier. The figure is best viewed in color/on screen.

- We propose a new architecture, the NFC layer for GCN-based models. It performs a convolution operation on the feature map constructed by the central node and selected neighbors, and can assign different weights to different features with the learned parameters in convolution kernels, which enables both the *node-level and feature-level attention* of the local neighborhood information in the learning process.

3.2 Methodology

3.2.1 Problem Definition

An undirected graph with N nodes can be represented as $\mathcal{G}$= $(\mathcal{V},\mathcal{E},\mathbf{X})$, where node $v_i \in \mathcal{V}$, edges $(v_i,v_j) \in \mathcal{E}$ $(i,j = 1,...,N)$, an adjacency matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$, and a feature matrix $\mathbf{X} \in \mathbb{R}^{N\times D}$ containing N D-dimensional feature vectors. A hidden representation of node v_i learned by the k-th layer of a model is denoted by $\mathbf{h}_i^{(k)} \in \mathbb{R}^{d_k}(d_k < D)$ and we initialize $\mathbf{h}_i^{(0)} = \mathbf{X}_i$.

Predictions on graphs are made by first embedding nodes $\mathbf{X}$ into a low-dimensional space $\mathbf{H}$, which is used for down-stream tasks, such as node classification, graph classification.

This chapter shows the proposed new GNN method called Node-Feature Convolution for Graph Convolutional Network (NFC-GCN) that applies the neighborhood sampling strategy and allows for feature-level attention during the neighborhood aggregation process. It mainly consists three steps as shown in Fig. 3.2: (1) we first select a fixed-size set of neighbors (*Neighbor Selection*) according to the similarity between the feature vectors of the given node and its neighbors to construct a fixed-size feature map; (2) subsequently, we introduce a convolutional layer (*Node-Feature Convolution (NFC)*), to learn a first-level representation by assigning different weights to node features; (3) finally, we feed the output of the NFC layer to a *Standard GCN* to obtain a second-level node representation.

3.2.2 Neighbor Selection and Ordering

To deal with varying node degrees in a graph, we select the most useful neighbors to obtain a feature map with fixed size. In this chapter, we focus on the node classification task, therefore we assume the useful neighbours should have relatively high similarity scores with the central node. For example, if two documents describe similar topics (belong to the same class and contain many similar keywords), the two documents are similar and their similarity measure should be high. As for ordering, nodes can be ordered using common node centrality metrics such as node degree, betweenness centrality [143], eigenvector centrality and PageRank. Previous [146, 92] works mainly focus on selecting or ordering nodes without considering node features.

We perform neighbor selection and ordering based on the node features. A variety of distance and similarity measures can be applied to measure the similarity between the central node and its neighbors, such as euclidean distance and cosine similarity. Euclidean distance would include difference in magnitude. The euclidean distance is often not a desirable metric for high-dimensional data mining applications, because of the concentration of distance in high-dimensional spaces, the ratio of the distances of the nearest and farthest neighbors to a given target is almost one [173, 2, 172, 32]. Since node feature vectors in citation graphs usually represent textual information often as bag-of-words, i.e., sparse vectors of weighted word frequencies in a document, which is often high-dimensional and sparse [163]. The

cosine similarity looks at "directional similarity" rather than magnitude differences, and is better at catching the semantic of each text, the direction the text points can be thought as its meaning so texts with similar meanings will be similar. Therefore, we perform neighbor selection and ordering based on the cosine similarity that has been commonly applied to measure document similarities [10, 172, 122, 186, 188]. The cosine similarity between the central node v_i and its neighbors v_j:

$$sim_{ij} = \frac{\mathbf{x}_i \cdot \mathbf{x}_j}{\|\mathbf{x}_i\| \|\mathbf{x}_j\|}, \tag{3.1}$$

where $\mathbf{x}_i \in \mathbb{R}^D$ and $\mathbf{x}_j \in \mathbb{R}^D$ are the feature vectors of v_i and v_j respectively. By specifying a hyperparameter *feature map bandwidth n*, we select the top $n-1$ neighbors with the highest similarity with the central node.

In practice, sparsely connected nodes may have less than $n-1$ neighbors. In this case, we select from the central node and all its neighbors based on probabilities *proportional* to the similarity sim_{ij} to get the desired feature map bandwidth n. For each node v_i, we obtain a **local feature map** $\mathbf{X}_i^{'} \in \mathbb{R}^{D \times n}$

$$\mathbf{X}_i^{'} = \left\{ \mathbf{x}_i, \left\{ \mathbf{x}_{j'}, j' \in \mathcal{N}'_i \right\} \right\}, \tag{3.2}$$

consisting of the feature vectors of the given node i and its selected neighbors $j' \in \mathcal{N}'_i$, where $\mathcal{N}'_i$ represents the selected neighbors of node v_i. This feature map can be seen from two dimensions: (1) the first dimension represents D node features, e.g., bag-of-words features with a fixed order for citation networks; (2) the second dimension represents the n nodes, including the central node and the $n-1$ selected neighbors. These nodes are ordered according to the neighbors' feature similarity with the central node from high to low.

Selecting a fixed number of neighbors can prevent neighborhood explosion and central node being "washed out". According to [216], the influence distribution of v_j on v_i can show how much a change in a neighbor v_j affects the final representation of the central node v_i in the last layer.

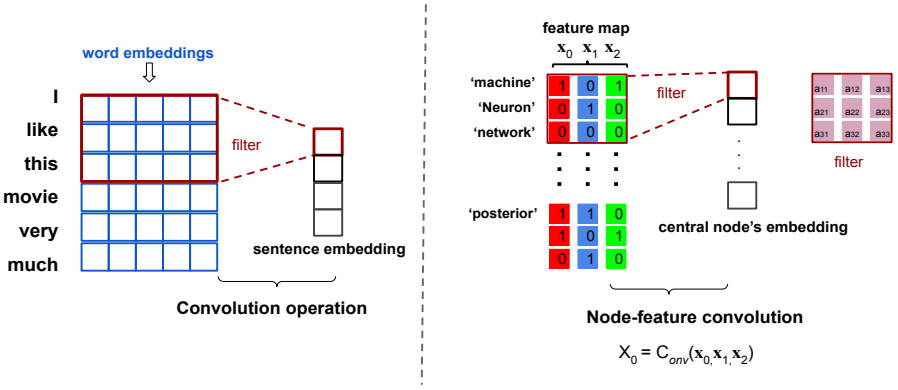


Fig. 3.3 Convolution on sentence and node feature map. The node feature corresponds to the word list, and the number of neighbors corresponds to the dimension of each word vector.

The *influence score and distribution* definition in [216] states that for a graph $\mathcal{G}=(\mathcal{V},\mathcal{E},\mathbf{X})$, $\mathbf{h}_i^{(0)}$ is the input feature and $\mathbf{h}_i^{(l)}$ is the learned hidden feature of v_i at the l-th layer (Eq. (2.9)). The influence score $I(i,j)$ of v_i by any v_j is the sum of the absolute values of the entries of the Jacobian matrix $\frac{\partial \mathbf{h}_i^{(l)}}{\partial \mathbf{h}_j^{(0)}}$. After $\mathcal{O}(log\,|N|)$ iterations of neighborhood aggregation using all neighbors, the representation of each node is "influenced almost equally by any other node" [81]. Thus, the final node representation captures mainly the global graph, with limited information about individual nodes. Performing neighbor selection for each GCN layer can alleviate the neighborhood explosion.

3.2.3 Node-Feature Convolution Layer

As a representation learning method, CNN works on fixed-size grids (e.g., images) or sequences (e.g., sentences) to tackle various problems such as image classification [220], machine translation [12], text or sentence classification [101] successfully. The *Neighbor Selection and Ordering* step enables us to apply a convolution operation on the node-feature map $\mathbf{X}_i'$ obtained.

In the citation graph, nodes usually represent documents, edges represent the citation links between documents, and node features represent textual information often as bag-of-words. The 0/1-valued feature vector of a node corresponds to an ordered word list from a dictionary, which is analogous to the ordered words in a sentence or document of an NLP task

[101]. For example, a part of the ordered word list of Cora includes “Machine”, “Markov”, “Monte-Carlo”, “Neural”, “Network”, then the local feature pattern “11100” could be highly related to the category of ’Reinforcement Learning’ and “10011” could be related to the category of “Neural Networks”. The adjacent features in a node feature vector are related. Therefore, we can use convolutions to find the potential local patterns that indicate a category.

As shown in Fig. 3.3, a fixed-size convolutional kernel scans over ordered words to obtain the representation of a sentence. Each 0/1-valued feature of our citation datasets (e.g. Cora, Citeseer) indicates the absence/presence of a corresponding word from a dictionary, which naturally inspires us to use 1-D convolution to scan over the feature vector of a node.

We perform convolution with C filters of size k and stride s on the local feature map $\mathbf{X}_i^{'} \in \mathbb{R}^{D \times n}$ of each node as

$$\vec{\mathbf{X}}_i = C_{onv}(\mathbf{X}_i^{'}). \tag{3.3}$$

The number of input channels is n. The output $\vec{\mathbf{X}}_i$ is of dimension $D' \times C$, where D' is determined by k, s, and C, the hyperparameters of NFC. Then, we flatten the output as following:

$$\mathbf{h}_i^{(0)} = flatten(\vec{\mathbf{X}}_i), \tag{3.4}$$

which is the first-level node representation.

3.2.4 Graph Convolutional Layer

Nodes with sparse connectivity (few first-order neighbors) may have insufficient information and need higher-order neighbors’ information to obtain better representations. Better representation of a given node can be obtained by considering L-order neighbors, where the best value for L depends on the data. Therefore, we pass the NFC representation through L additional GCN layer(s) to enable a central node aggregating information from higher-order neighbors. An aggregation operator works on the first-level node representation (Eq. (3.4)) to learn another new representation of node i, as in Eq. (2.9).

After L GCN layers, the final representation $\mathbf{h}_i^{(L)}$ will be passed to a fully-connected layer with a $softmax$ activation function. For multi-class classification, the loss function is defined

Algorithm 3.1 NFC-GCN

Input: $\mathcal{G}$= $(\mathcal{V}, \mathcal{E}, \mathbf{X})$ with N nodes;

Adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$;
Feature matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$;
Labeled nodes $\mathcal{V}_l$;
Label indicator matrix $\mathbf{Y}_{lf} \in \mathbb{R}^{|\mathcal{V}_l| \times F}$;
The number of selected neighbors is (n-1);
The parameters in the node-feature convolution process: filter size k, stride s, the number of filters: C, the convolution operation $C_{onv}(\cdot)$.
for each $v_i \in \mathcal{V}_l$ **do**
    if $d_i > n-1$ **then**
        choose $n-1$ neighbors according to similarity from high to low
    else if **then**
        select $(n-1-d_i)$ nodes based on probabilities proportional to the similarity
    end if
    feature map construction: $\mathbf{X'}_i = \left\{\mathbf{x}_i, \mathbf{x}_{j'}, j' \in \mathcal{N'}_i\right\}_n$
    node feature convolution: $\vec{\mathbf{X}}_i = C_{onv}(\mathbf{X'}_i)$
    flatten the embeddings: $\mathbf{h}_i^{(0)} = flatten(\vec{\mathbf{X}}_i)$
    for each layer l, l=1,...,L **do**
        feature transformation: $\mathbf{h}_i^{(l)} = \sigma(\mathbf{W}^{(l)}(\mathbf{h}_i^{(l-1)} + \sum_{j \in \mathcal{N}_i} \mathbf{h}_j^{(l-1)}))$;
    end for
end for

as the cross-entropy error over all labeled examples:

$$\mathcal{L} = -\sum_{l \in \mathcal{V}_l} \sum_{f=1}^{F} \mathbf{Y}_{lf} \ln \mathbf{h}_l^{(L)}, \tag{3.5}$$

where $\mathcal{V}_l$ is the set of node indices that have labels and F is the dimension of output features equaling to the number of classes. $\mathbf{Y}_{lf} \in \mathbb{R}^{|\mathcal{V}_l| \times F}$ is a label indicator matrix. Algorithm 5.1 summarizes the general framework of NFC-GCN.

Figure 3.4 demonstrates the neighbor selection achieved via different models (GCN-GCN, NFC-GCN, NFC-NFC). Considering a two-layer GCN (i.e., GCN-GCN) in Fig. 3.4(b), after

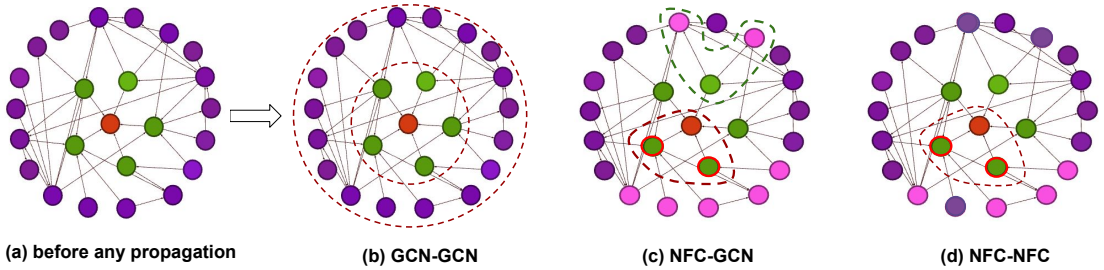


Fig. 3.4 Comparison with different layers. Take the red node as an example, the red node's first-order and second-order neighbors are respectively green and purple nodes, as shown on the left. After a two-layer GCN, the central node contains information from all first-order neighbors and second-order neighbors as shown in (b) GCN-GCN. After one NFC and one GCN layer, each node contains information from all its first-order (directly) and part of its second-order neighbors' information (indirectly) as in (c) NFC-GCN. After two NFC layers ((d) NFC-NFC), the central node only contains the two most similar first-order neighbors and part (less than in NFC-GCN) of second-order neighbors' information.

the first propagation (first GCN layer), each node (e.g., the red node) only contains the first-order neighbors' information (green nodes). After the second propagation (second GCN layers), each node aggregates information from all its first-order and second-order neighbors (purple nodes). Figure 3.4 (c) shows NFC with one GCN layer (i.e., NFC-GCN). In the first propagation, each node (the red node) only aggregates information of the top two most similar first-order neighbors (two green nodes with red circles). After the first propagation (NFC layer), each green node's representation also contains information from its two most similar neighbors as well (the dash green curve in (c) NFC-GCN). After the second GCN layer, the central node's representation contains information from all the first-order neighbors (green nodes) but only part of (selected) second-order neighbors (pink nodes). Therefore, NFC-GCN only selects part of the neighbors for the central node's representation learning even after adding GCN layers. Compared with NFC-GCN, each node contains less information from its first-order and second-order neighbors after two NFC layers. Because the central node (red node) aggregates two green nodes with the red circle, each green node contains information of its two most related neighbors (pink nodes) after the first NFC layer. In the second propagation (after the second NFC layer), the central node still aggregates two first-order neighbors who already contain two of their own first-order neighbors' (pink nodes)

information. Therefore, after two NFC layers, each node contains two first-order neighbors and part of second-order neighbors information (indirectly).

Compared to the standard GCN model (GCN-GCN - $\cdots$), NFC-GCN and NFC-NFC can alleviate the neighborhood explosion (over smoothing problem) and help avoid the central node being "washed out" due to aggregating too many neighbors. A pure NFC model (NFC-NFC-$\cdots$) can reduce the considered neighbors further. But how much information (the number of neighbors) should be considered for the best representation learning of a given node has no precise answer.

3.2.5 Computational Complexity

A key part in our method is the NFC-layer, and the filters are shared by all nodes in a graph. Therefore, the computation of the parameters in the filters can be parallelized across all nodes. The computational complexity of a GCN layer as shown in Eq. (2.9) is $\mathcal{O}(N \times D_{l-1} \times D_l)$, while an NFC-layer (Eq. (3.3)) is $\mathcal{O}(N \times C \times k \times n)$. In GCN, the models' complexity is related to the node feature dimension ($\mathbf{W}^{(0)} \in \mathbb{R}^{D_1 \times D}$) and this may lead to many parameters in the model if the dimension of the original node feature is high. It should be emphasized that the NFC layer inherits the advantage of CNN whose parameters are not related to the image size, so the complexity of NFC layer is not influenced by the node feature (C, k, n are hyperparameters).

3.2.6 Differences with Existing GNNs

Sampling-based methods. Our method selects the neighbors according to their similarities with the central node from high to low. In contrast, GraphSAGE [72] and DropEdge [156] select the neighbors randomly with random work or drop out some edges in a graph. FastGCN [36] directly sampled the nodes in each hidden layer independently and JP-networks [216] sampled learned intermediate representations for a given node to get the final node representation. Graph DropConnect (GDC) [75] and GeniePath [124] automatically learn the connections in a graph.

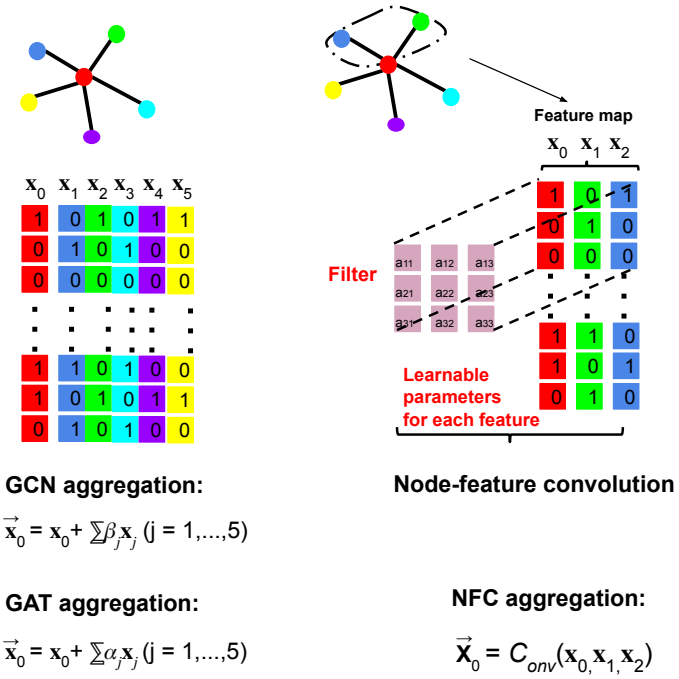


Fig. 3.5 Differences between GCN, GAT, and NFC-GCN. In the aggregation process, both GCN and GAT aggregate all the neighbors with different weights. The weights β_j, for each neighbor related to node degree are fixed in GCN. While α_j is learnable in GAT. But all the features in each feature vector share the same weights β_j, or $\alpha_i, i, j \in (1,5)$. In contrast, our method performs convolution operation on the selected node-feature map to assign different weights (such as $a_{11}, a_{12}, ..., a_{33}$) to different features in different neighbors.

Neighbor weighting-based methods. In DisenGCN and GAT, all the features within a feature vector still shared the same weight. In contrast, our method selects part of the neighbors and learns to assign different weights to different features in different neighbors. Masked GCN, GNN-Film can do a feature-level attention, but they consider all neighbors in the learning process, which may introduce much noisy information. Besides, the complexity of their models is related to the node feature, which can be time-consuming for high-dimension input. Our model only selects the most related neighbors before aggregation and applies CNN in the aggregation. Our model inherits the advantage of CNN and its model size is not related to the input data. LGCL used CNN after a GCN layer, so it inherited the limitations of GCN as we mentioned above. In contrast, our method uses CNNs in the first step to extract useful information from raw node features. Besides, LGCL constructed the feature map by selecting the k-largest values for each feature from *all*

Table 3.1 Overview of the three datasets with standard splits as in the Fast-GCN [36] (Val. means Validation).

Dataset	Nodes	Edges	Features	Classes	Train/Val./Test
Cora	2,708	5,429	1,433	7	1,208/500/1,000
Citeseer	3,327	4,732	3,703	6	1,827/500/1,000
PubMed	19,717	44,338	500	3	18,217/500/1,000

neighbors' embeddings (learned from one GCN layer), which broke the correspondence in the original node embedding. In our method, the feature map is constructed from the central node and neighbors' (selected and ordered) raw node features. Therefore, the constructed feature map has a consistent structure, which is suitable for CNNs to perform on.

3.3 Experiments

In this section, we conduct extensive experiments on three real-world benchmark datasets to evaluate our proposed model from three main aspects: 1) the node classification performance in terms of accuracy and convergence; 2) the effectiveness of NFC aggregation; 3) parameter sensitivity. Finally, we analyze the limitations and advantages of our method.

3.3.1 Datasets

We use three citation network benchmark datasets, Cora, Citeseer and PubMed, that have been widely used in previous related work. We use the same train/validation/test splits as in [36]. Table 3.1 shows an overview of the datasets.

- **Cora.** The Cora dataset contains 2,708 documents (nodes) classified into seven classes (i.e., Neural Networks, Rule Learning, Probabilistic Methods, ..., Reinforcement Learning) and 5,429 citation links (edges). We treat the citation links as (undirected) edges and construct a binary, symmetric adjacency matrix. Each document has a 1,433 dimensional sparse bag-of-word feature vector and a class label.

- **Citeseer.** The Citeseer dataset contains 3,327 documents classified into six classes (i.e., Agents, AI, ..., ML) and 4,732 links. Each document has a 3,703 dimensional sparse bag-of-word feature vector and a class label.

- **PubMed.** The PubMed dataset contains 19,717 documents classified into three classes (Diabetes Mellitus Type Experimental, Diabetes Mellitus Type 1, Diabetes Mellitus Type 1) and 44,338 links. Each document has a 500 dimensional sparse bag-of-word feature vector and a class label.

3.3.2 Baselines

We compare NFC-GCN against nine competing methods in total. We consider four representative non-GCN methods: Locally Linear Embedding (LLE) [157], Laplacian Eigenmaps (LE) [16], Graph Factorization (GF) [5] and DeepWalk [151] that only utilise graph structure in the node representation learning. We use the implementation provided by authors of [69] with standard settings used in their paper. In order to ensure the baselines have sufficient diversity, we compare against seven state-of-the-art models: GCN,[1] two sampling based methods (FastGCN,[2] GraphSAGE[3]) and four neighbor weighting-based methods (Disen-GCN,[4] GAT,[5] LGCL[6]). We all use the publicly available implementation and report the mean accuracy of 100 runs with random weight initializations.

- **GCN.** Graph Convolutional Networks [103] is the standard baseline. In our experiments, we use a two-layer GCN model. For the key hyperparameters, we swept the number of hidden units in the set {16, 32, 64, 128}, L2 regularization $\{5\times10^{-3}, 5\times10^{-4}, 5\times10^{-5}\}$, dropout rate {0.2, 0.4, 0.6}, learning rate {0.01, 0.001, 0.0001}. We set the max training epoch to 1000 and early stopping to 10.

[1] https://github.com/tkipf/gcn
[2] https://github.com/matenure/FastGCN
[3] https://github.com/williamleif/GraphSAGE
[4] https://jianxinma.github.io/disentangle-recsys.html
[5] https://github.com/PetarV-/GAT
[6] https://github.com/HongyangGao/LGCN

- **Sampling-based methods.** We compare our method with FastGCN [36] and GraphSAGE [72]. We split the train/validation/test as [36], so we use the same hyperparameters as in their paper. For FastGCN, we use two hidden layers, the batch size is 256 for Cora, Citeseer and 1024 for PubMed, the sample sizes are 400, 400 and 100 for Cora, Citeseer and PubMed, the learning rate is 0.001, and dropout is set as zero. For GraphSAGE, we apply the mean aggregator (GraphSAGE-mean usually gets the best results) and use two layers with neighborhood sample sizes 25 (for the first layer) and 10 (for the second), and the batch size is the same with FastGCN.

- **Neighbor weighting-based methods**. For these methods, we swept the common key hyperparameters: hidden units, L2 regularization, dropout rate and learning rate, as in GCN. Graph attention networks [191] learns to assign different weights to different neighbors. In the experiment, we apply two GAT layers and the number of attention heads are in the set {2,4,8}. Learnable Graph Convolutional Layer (LGCL) [58] performs convolution on the reconstructed node embeddings after one GCN layer. K ranges between {8,16,32} for the K-component feature vectors and dropout $\in$ {0.2, 0.4, 0.6, 0.8 } is applied on both input feature vectors and adjacency matrices in LGCN. We apply two LGCL layers for Cora and Citeseer, one LGCL layer for PubMed and stop the training within 10000 epochs. DisenGCN [129] can treat different cluster of neighbors differently and we set the number of channels $\in$ {4,8,16,32}. The max training epoch is set to 1000 for DisenGCN.

3.3.3 Hyperparameters Setting

Hyperparameters for NFC-GCN. We swept the common key hyperparameters: hidden units, L2 regularization, dropout rate and learning rate, as in GCN. Other key hyperparameters for NFC layer are set as: the number of neighbors $\in$ {1,2,3,4,5}, the filter size $\in$ {32, 64, 128}, the number of filters $\in$ {8,16,32, 64}, the number of stride $\in$ {16, 32, 64}. We employ the early stopping strategy based on the validation accuracy and train 200 epochs at most. As mentioned earlier, Compared with GCN layer, NFC layer is a more powerful tool, but it

Table 3.2 Node classification accuracy (%) (mean ± 95% confidence interval over 100 runs). (**Best**; Second best)

Methods	Cora	Citeseer	PubMed
LLE	30.5 ± 0.390	20.5 ± 0.137	39.8 ± 0.190
LE	29.6 ± 0.980	21.2 ± 0.190	39.8 ± 0.109
GF	30.7 ± 0.180	20.9 ± 0.590	39.9 ± 0.190
DeepWalk	55.2 ± 0.157	44.1 ± 0.980	77.6 ± 0.590
GCN	88.1 ± 0.235	77.8 ± 0.216	86.8 ± 0.176
Fast-GCN	85.0 ± 0.470	77.6 ± 0.1235	88.0 ± 0.627
GraphSAGE	82.2 ± 0.135	71.4 ± 0.165	87.1 ± 0.921
GAT	80.4 ± 0.255	75.7 ± 0.431	85.0 ± 0.390
LGCL	86.9 ± 0.216	77.5 ± 0.392	84.1 ± 0.118
DisenGCN	87.4 ± 0.333	77.0 ± 0.588	87.2 ± 0.400
NFC-GCN	88.7 ± 0.255	**79.4 ± 0.431**	89.7 ± 0.137
NFC-NFC	**89.6 ± 0.274**	78.9 ± 0.470	**90.4 ± 0.157**

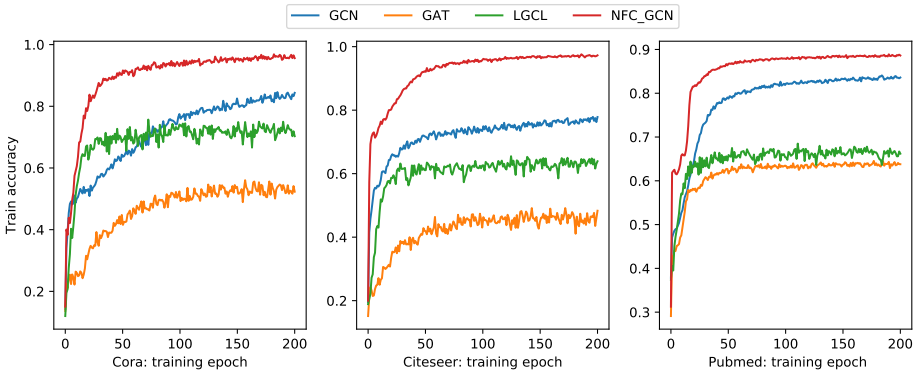


Fig. 3.6 Comparison of training accuracy with respect to the training epochs.

is more time consuming when the number of filters or neighbors is high. Considering both accuracy and efficiency, we use different combinations: NFC-GCN, NFC-NFC to learn the final node embeddings. Our code are available online. [7]

3.3.4 Performance for Node Classification

In this section, we compare the node classification accuracy with baselines, besides we compare the convergence with three mostly related methods: GCN, GAT, and LGCL.

[7] https://github.com/LiZhang-github/NFC-GCN

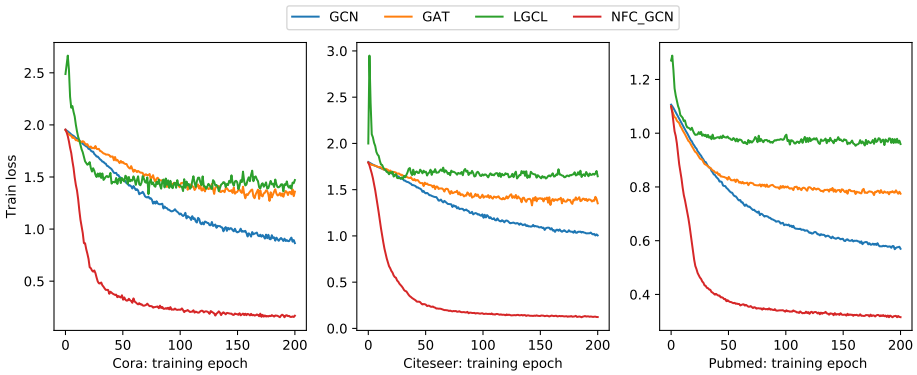


Fig. 3.7 Comparison of training loss with respect to the training epochs.

- **Node classification.** Results in classification accuracy are summarized in **Table 3.2**. For the first four non-GCN methods, they do not perform well for they only utilise the structure information. FastGCN and GraphSAGE focus on improving the training efficiency so they have slightly poorer results than GCN. GAT utilize neural networks to learn attention scores for a given node's neighbors. It is time consuming especially for Cora and Citeseer with high-dimensional node features. DisenGCN applies neighborhood routing mechanism to cluster a given node's neighbors, which is more suitable for dense graphs. While Cora, Citeseer and PubMed are relatively sparse and their median node degrees are 4, 3, 3 respectively. LGCL can get competitive results on Cora and Citeseer, while it does not perform well on PubMed. Two possible reasons are 1) it inherits the limitation of GCN and 2) it reorganises the original embedding in the process of constructing feature maps as mentioned in Section 3.2.6.

  Our models NFC-GCN and NFC-NFC achieved state-of-the-art performance across all the datasets. This suggests that applying the NFC layer to work on the most related and fixed-size neighbors can be beneficial for learning node representation. NFC-GCN can aggregate more neighborhood information, and it may be more suitable for sparse graph (NFC-GCN gets better result on Citeseer.) NFC-NFC can further alleviate the over-smoothing problem and may be more suitable for dense graph. Considering efficiency, NFC-GCN is faster. For Cora, Citeseer and PubMed, times for each training epoch are 9, 9.4, and 14 seconds respectively, while they are around 18, 18 and 28

seconds in NFC-NFC. We can flexibly combine NFC layer and GCN layer, depending on the requirement of downstream tasks.

In addition, our method achieves better performance in fewer training epochs (less than 100 epoch) on all the datasets. However, we should note that the training time per epoch for our method is more than GCN, GAT, Fast-GCN, possibly due to the larger number of parameters in our model. In future, we can investigate ways to improve the per-epoch computational efficiency.

- **Accuracy, loss over training epochs. Figures 3.6** and **3.7** show how the training accuracy, training loss change with respect to the number of training epochs. We do not use early stopping in our model for a better comparison with GCN, GAT and LGCL. Our method achieves a good performance in a few training epochs, while GCN, GAT and LGCL need more than a hundred training epochs. Moreover, training accuracy/loss of NFC-GCN change in a more stable way with the same optimization parameters on the same training data as GCN, GAT and LGCL. This confirms that the first-level node representation learned from the node-feature convolution improves the subsequent classification tasks.

On the whole, Table 6.2, Figs. 3.6 and 3.7 show that our method has a better performance on both node classification accuracy and convergence.

3.3.5 Effectiveness of NFC aggregation

We further present studies of three different aggregation methods: GCN aggregation, GAT aggregation, and NFC aggregation. And we also illustrate the effectiveness of CNNs on the ordered node features.

To show the effectiveness of NFC (our key contribution), we compare GCN and GAT aggregation methods with NFC using a fixed-size set of neighbors.[8] Then we feed the aggregated representation of each node to a classifier directly without adding additional GCN layers. We carry out this experiment over all datasets and choose 5 neighbors for each

[8]LGCL uses GCN aggregation over the raw node features while its first layer is the same as GCN.

Table 3.3 Node classification accuracy for different aggregation methods with five neighbors and only one aggregation step (%) (mean ± 95% confidence interval over 100 runs).

Aggregation	Cora	Citeseer	PubMed
GCN	64.8 ± 0.2918	74.1 ± 0.5939	80.0 ± 0.1626
GAT	64.2 ± 0.4508	74.2 ± 0.4508	82.2 ± 0.6272
NFC	**86.0 ± 0.1705**	**78.9 ± 0.3175**	**89.7 ± 0.1155**
Improvement	21.2	4.8	7.5

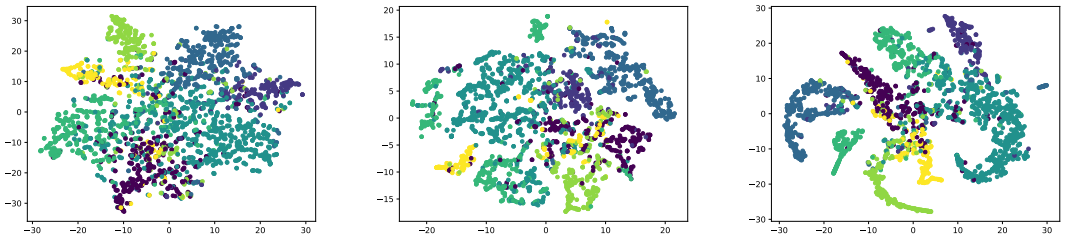
Table 3.4 The effectiveness of NFC aggregation. Cen: central node (without convolution operation); Cov(C): central node with convolution operation; Cov(CN): node-feature map (containing central node and neighbors' features) with convolution operation (%) (mean ± 95% confidence interval over 100 runs).

Dataset	Cen	Cov(C)	Cov(CN)
Cora	54.7 ± 1.9180	72.9 ± 1.0035	86.0 ± 0.1705
Citeseer	69.9 ± 2.2530	73.1 ± 1.5915	78.9 ± 0.3175
PubMed	80.9 ± 0.2918	85.7 ± 1.0038	89.7 ± 0.1155

aggregation methods. The results are summarized in **Table 3.3**. Our method increases the testing accuracy greatly over GCN/GAT, demonstrating that NFC-based aggregation can learn a more effective node representation for subsequent tasks. It should also be emphasized that only the NFC can achieve competitive performance without additional GCN layers.

Besides the quantitative evaluation, we also investigate the effectiveness of different aggregation methods qualitatively. We provide t-SNE [130] visualizations to map the embeddings obtained from GCN, GAT and NFC aggregation on the Cora dataset in 2D space. In **Fig. 3.8**, all the embeddings exhibits discernible clustering in the projected 2D space. The GAT visualization is poorer than the GCN visualization, which is consistent with the results in Table 3.3. NFC aggregation obtains the best visualization with nodes clustered into the most compact clusters.

A further evaluation is conducted to illustrate how NFC works on node-feature maps. We feed the central node's feature vector to the classifier directly (Cen), which is treated as a baseline. For comparison, we first perform a convolutional operation only on the given node's feature vector (one channel) and feed the new representation to the classifier (Conv(C)). This



(a) GCN aggregation (b) GAT aggregation (c) NFC aggregation

Fig. 3.8 Visualization of the embeddings on the Cora dataset. We map the embeddings learned from GCN, GAT and NFC aggregation to the 2-D space using t-SNE. Node colors denote classes.

Table 3.5 Node degree statistics.

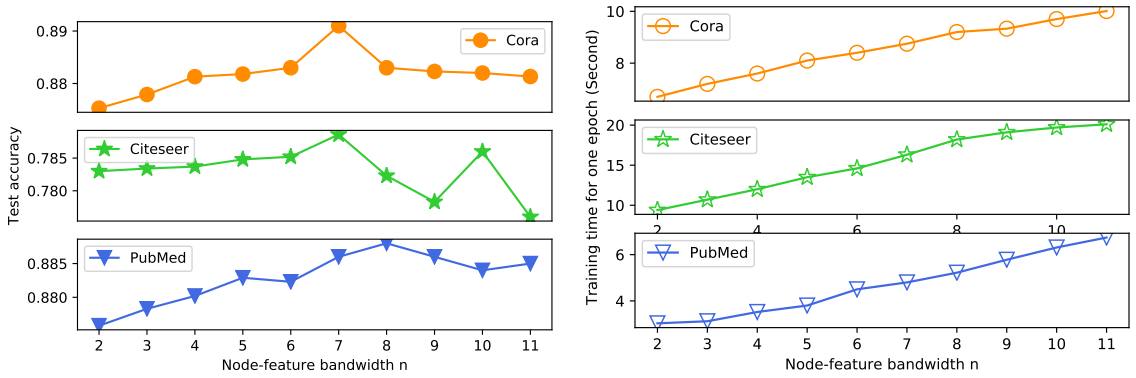
Dataset	Highest	Lowest	Average	Median
Cora	168	1	4.9	4
Citeseer	99	1	3.7	3
PubMed	171	1	5.5	3

is used to illustrate how the convolution works on the node feature dimension. Next, we perform a NFC operation on the feature map comprised of the given node and five neighbors (six channels), and feed the new representation to the classifier (Conv(CN)). The experimental results are shown in **Table 3.4**. The main finding is that the NFC not only operates effectively on the node feature dimension, but also extracts more useful information from different channels in the node-feature map.

This study shows that the advantage of NFC (aggregation assigning different weights to different features for different neighbors), as shown in Table 3.3 and Fig 3.8. Besides, applying the convolution operation on the ordered features and including neighbors information can both benefit to the central node's representation learning, as shown in Table 3.4.

3.3.6 Node Bandwidth Study

We first study the effect of varying the node bandwidth n in the node-feature convolution process. **Table 3.5** shows the distribution of node degrees over the three datasets. The node degree varies from one to 171, which illustrates that there is a need to select the neighbors



(a) Test accuracy.

(b) Time per epoch.

Fig. 3.9 Effect of node bandwidth n on accuracy and time per epoch.

for a given node. The average node degree is three or four, and we vary n from two to 11. The results in **Fig 3.9 (a)** show consistent improvement in accuracy with increasing n from two to seven. A higher n implies more feature diversity, and this can be especially helpful for the representation learning of nodes with sparse connectivity and features. However, for n greater than seven, performance drops. A possible explanation is that aggregating too many neighbors has a negative influence (the central node's own information being washed away) on the given node's representation learning. Note that the computation time per training epoch also increases with n, therefore, there is a trade-off between the classification accuracy and computational time when choosing n.

3.3.7 Model Depth Study

We study the influence of model depth (number of GCN layers) on classification performance in the data splits in [103]. We change the GCN layers from one to five and the results are summarised in **Fig. 3.10**. Our method is less sensitive to the number of hidden layers. This indicates that the NFC-GCN representation is more robust to model depth. It performs well even when a higher-order neighborhood is considered. Note that the best test accuracy for our method is not better than GCN on PubMed. One possible reason is that this data splits in [103] have only 60 labeled training nodes that is too small for NFC-GCN. Another possible reason is that we did not tune NFC-GCN hyperparameters (set as 3.3.3) for different layers.

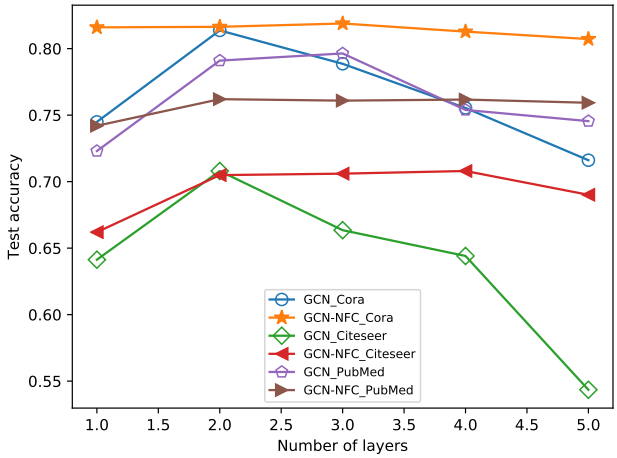


Fig. 3.10 Performance comparison on deeper models. On the Cora, Citeseer and PubMed datasets, we employ the same experimental setups and increase layers of GCN and NFC-GCN to up to five. GCN-NFC has a better overal performance for deeper models and its test accuracy is more steady than GCN when we increase the number of layers.

With further tuning and optimisation, NFC-GCN has the potential to get better results in classification accuracy for deeper models.

3.3.8 Discussion

NFC-GCN embodies the ideas from GCN and its extensions such as sampling-based methods and GAT. NFC-GCN applies convolution layer on a fixed-size node-feature map to assign different weights to different features in different neighbors. In the following, we discuss the limitations and new opportunities for this new architecture.

- **Limitations.** The main limitation is that NFC-GCN requires more training samples and the computation cost for each epoch is higher than GCN. Nonetheless, deep learning models are known to work well on larger datasets and be computationally expensive. One future work will aim to improve the performance of our method on the small training datasets and also the computational efficiency. The key step of our model is neighbour selection. We select the neighbours based on the assumption that useful neighbours should have a relatively high similarity score with the central node for the

node classification task. While this assumption may not be suitable for other tasks, such as link prediction, and network reconstruction. Two nodes can be linked even if they have a lower similarity score, for example, papers (nodes) from the medical domain can cite (connect) papers (nodes) from the computer science domain, which would be conflict with the assumption.

- **Representation learning ability.** When we only use one NFC layer to learn nodes' new representations and feed them directly to a classifier layer, we can obtain a competitive performance compared to GCN and its extensions. One potential future direction is to get rid of the GCN layer totally.

- **New architectures and deeper models.** The NFC layer can be used in conjunction with any of the competing methods (not only GCN). In particular, NFC-GCN allows for a deeper model, and has the potential to get better classification accuracy with well-tuned hyperparameters. Furthermore, deeper models can enable other powerful machine learning techniques to be better applied to graphs, such as transfer learning.

3.4 Summary

In this chapter, we proposed a novel model: Node-Feature Convolution for Graph Convolutional Network (NFC-GCN) to solve the first research question: node-level and feature-level attention of local neighborhood information. We constructed a new node-feature convolutional (NFC) layer to work on a fixed-size feature map that contains features from selected neighbors, and the learned parameter of the convolutional kernels can assign different weights to different features, which allows the proposed model can consider both node-level and feature-level attention of local neighborhood information in the learning process. Meanwhile, the neighbor selection step can also alleviate neighborhood explosion problem. Thus NFC-GCN addresses the two main limitations of existing GCN models. Experimental results showed that NFC-NFC outperforms current competing GCNs on three benchmark datasets in node classification. In addition, the NFC layer can be a plug-in module and integrated with

other GCN extensions, e.g., FastGCN [36], jumping knowledge networks [216], GMWW [154] and MixHop [1], enabling these methods to have a feature-level attention. However, the number of neighbors varies greatly and the NFC layer can not operate on the graphs directly, the neighbor selection and ordering step would be time consuming, especially for large-size graphs, which is the main limitation of NFC-GCN.

Chapter 4

Learnable Aggregator for Graph Convolutional Network

4.1 Introduction

In Chapter 3, NFC-GCN considers feature-level attention of the local neighborhood information in the learning process, but it can not directly operate on graphs due to the irregular connectivity and lack of orderliness of nodes in a graph. In this chapter, we extend NFC-GCN to a more flexible and general framework by lifting constraints on the input data format

As pointed in Sec. 2.3.3 and Sec. 3.1, edges in real graphs are often noisy or defined via user-defined thresholds [105]. Therefore, edges may not clearly correspond to label agreement uniformly across the graph [174]. In Cora, Citeseer and PubMed, 19%, 26%, and 20% of the edges, respectively, connect with nodes from different classes. Besides, each feature within a neighbor feature vector may play a different role for the central node's representation learning [243, 111, 73]. In such cases, mean, pooling, or sum aggregators are not optimal choices in learning useful representations from the noisy neighborhood of the central node. It is necessary to filter both node and feature information before aggregation, especially for graphs with node features.

There are also *learnable* aggregators proposed to automatically filter the neighborhood information. Graph Attention Network (GAT) borrows the idea of attention mechanisms

[190] to learn to assign different weights to different neighbors in aggregation [191]. However, all individual features in a feature vector are treated equally. Learnable Graph Convolutional Layer (LGCL) performs convolution operation in the aggregation process, which can assign different weights to different features [58]. But, using regular convolution operation on graphs requires the number of neighboring nodes for each node remains the same, and these neighboring nodes are ordered. LGCL transforms the graph into grid-like structure by selecting the top-d values in each feature dimension from all the neighbors. The covolutional operator mixes or reorganizes neighborhood information, which makes it difficult to interpret the learned representation because we can not distinguish which node and feature have a salient influence on the prediction result. MaskedGCN [219] only propagates a certain portion of each neighbor (v'_j) attributes (features) to the central node (v_i) via a mask vector, however the mask vector is learned independent to the central node v_i, and depends on the consistencies of each feature between v_j and v'_j neighbors. This mechanism, only considering the attribute distributions in local neighbourhoods, is still influenced by neighbors from different class with the central node. Thus, a more straight way is to model the v_i and v_j relationship directly to learn the mask indicator.

In this chapter, we aim to design a more adaptive and interpretable aggregator satisfying the following five **D**esirables.

- **D1 & D2:** To deal with graph structures, the aggregator should **1)** be able to handle variable-sized neighbors [191, 139], and **2)** be invariant to the ordering of neighbors [139]. Unlike images and sentences, graphs usually have no regular connectivity and neighboring nodes have no natural ordering.

- **D3 & D4:** To enhance the discriminating power, the aggregator should **3)** be discriminative to node-level and feature-level neighborhood information [191, 58], **4)** be able to discriminate graph structures in the embedding space [215]. Real-world graphs are noisy and the aggregator should automatically identify the important information from the neighborhood.

- **D5:** For practical applications where interpretability is needed, the aggregator should **5)** be able to explain learned representations in relation to the prediction and robust to structure and feature noise. Real-world data are often noisy so aggregating information from noisy graph structures and node features can cause significant difficulties in accurate prediction and useful interpretation [222]. An explainable and robust aggregator can increase the trustworthiness and real-world performance.

To this end, we unify current learnable aggregators in a general framework: learnable aggregator for GCN (LA-GCN). This framework introduces an auxiliary model that can extract customized high-level knowledge from a given node's neighbors to guide the aggregation process. Under this framework, we propose a new model called LA-GCN$_{\text{Mask}}$ consisting of a new aggregator function, *mask aggregator*, and a carefully designed auxiliary model shared by all nodes in a graph that satisfies **D1** and **D2**. Firstly, a given node and its neighbors are fed into the auxiliary model to get a specific mask for each neighbor. Then the mask aggregator performs a Hadamard product between the feature vector of each neighbor and its corresponding mask before aggregation. In this way, the mask aggregator can learn to assign different weights to different features in different neighbors, which leads to better discriminativeness to node and feature information (**D3**) and also enables better interpretation of the learned representation (**D5**). The proposed aggregator sums up all the filtered neighbors of the central node as its learned representation, meeting **D4**. Finally, we compare our proposed aggregator and the mentioned aggregators in Table 4.1 to show our novelty. Our aggregator satisfies all desideratas, enabling a leap in model capacity.

We evaluate LA-GCN$_{\text{Mask}}$ on three popular citation graphs and one large social graph for node classification, and three bioinformatics graphs for graph classification. Our results confirm that node-level and feature-level attention of neighborhood information in aggregation can lead to significant performance gains. In addition, we visualize the learned mask to show that it can identify important features and nodes, which provides an interpretable explanations for prediction. Finally, we study the robustness of our model on graphs with structure and node feature noise. In both structure-noisy and feature-noisy graphs, LA-GCN$_{\text{Mask}}$ consistently outperforms popular baselines (GCN, GAT and LGCL), with up to

Table 4.1 Comparisons of the traditional aggregators and our proposed aggregator. Outline of related work in term of fulfilled ($\surd$) and missing ($\times$) desirable characteristics (D3-n means node-level attention and D3-f means feature-level attention in Desirable 3).

Desirable	Mean	Mean$_w$	Sum	Sum$_{lw}$	*Conv*	Mask
D1	$\surd$	$\surd$	$\surd$	$\surd$	$\times$	$\surd$
D2	$\surd$	$\surd$	$\surd$	$\surd$	$\times$	$\surd$
D3-n	$\times$	$\surd$	$\times$	$\surd$	$\surd$	$\surd$
D3-f	$\times$	$\times$	$\times$	$\times$	$\surd$	$\surd$
D4	$\times$	$\times$	$\surd$	$\surd$	$\surd$	$\surd$
D5	$\times$	$\times$	$\times$	$\times$	$\times$	$\surd$

9.82% (Cora) and 15.05% (Citeseer) improvement on structure-noisy graphs and 10.67% (Cora) and 3.60% (Citeseer) improvement on node feature-noisy graphs, in terms of node classification accuracy.

In summary, our contributions of this chapter are threefold:

- We unify several existing methods in a LA-GCN framework and propose a new mask aggregator, a new attention mechanism allowing both node-level and feature-level attention.
- We comprehensively evaluate the superiority of the proposed LA-GCN$_{\text{Mask}}$ on seven graphs with different sizes and types for both node and graph classification tasks.
- We demonstrate that the proposed model can provide interpretable explanation for the prediction, also study the robustness of our model on both structure and node feature noisy graphs.

4.2 Methodology

This chapter unify current learnable aggregator in one framework: LA-GCN, that utilizes an auxiliary model to guide the aggregation process, which enables the aggregator to satisfy all the desirables. In this section, we first describe the framework, followed by theoretical

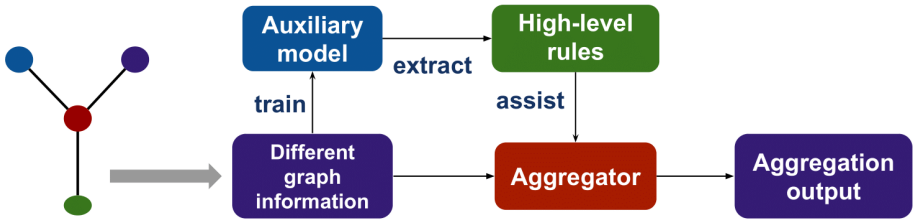


Fig. 4.1 LA-GCN framework. The key idea is to utilize an auxiliary model to assist the aggregator to deal with different neighborhood information in a customized schema.

motivation for our model: LA-GCN$_{\text{Mask}}$. Then, we compare our model with prominent GCN based methods.

4.2.1 Problem Definition

An undirected graph with N nodes can be represented as $\mathcal{G}$= $(\mathcal{V},\mathcal{E},\mathbf{X})$, where node $v_i \in \mathcal{V}$, edges $(v_i,v_j) \in \mathcal{E}$ $(i,j = 1,...,N)$, an adjacency matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$, and a feature matrix $\mathbf{X} \in \mathbb{R}^{N\times D}$ containing N D-dimensional feature vectors. A hidden representation of node v_i learned by the k-th layer of a model is denoted by $\mathbf{h}_i^{(k)} \in \mathbb{R}^{d_k}(d_k < D)$ and we initialize $\mathbf{h}_i^{(0)} = \mathbf{X}_i$.

Predictions on graphs are made by first embedding nodes $\mathbf{A}$ with $\mathbf{X}$into a low-dimensional space $\mathbf{H}$, which is used for down-stream tasks, such as node classification, graph classification.

4.2.2 Framework of LA-GCN

A key challenge is how to design an efficient aggregator that suits for each node in a graph since each node has different neighbors no matter the numbers or categories, and satisfies the mentioned desirables. Intuitively, this requires each node with a specific model, which is quite impossible, for real-world graphs can contain millions or billions nodes [70].

Inspired by the weight sharing property of CNNs [110] and attention mechanism [190], we use a shared auxiliary model to extract high-level knowledge or rules from the given graph information, and the learned rules are used to assist the aggregation process as shown in Fig 7.3. It is a flexible and general framework that can unify mentioned GAT [191] and LGCL [58], and we give a detail comparison in Section 4.2.7.

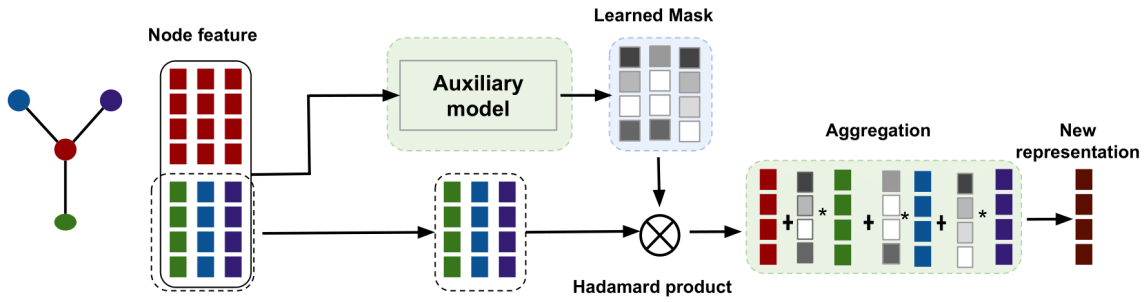


Fig. 4.2 LA-GCN$_{\text{Mask}}$ consists of three steps: 1) train an auxiliary model with a given node and the feature vectors of its neighbors; 2) generate the mask for each neighbor from the auxiliary model; 3) aggregate the neighbors (after multiplying the corresponding mask) to get a new representation of the central node.

4.2.3 Theoretical Studies of Aggregator

Under this framework, we carefully design our auxiliary model and propose a new aggregation function: mask aggregator. Our ultimate goal is to design an aggregator that can satisfy all the disirables. We start from the oretical study of the aggregator function, which enables the formulation of our aggregator that simultaneously satisfies our desirables. In this subsection, we mainly study the aggregator function from graph datasets's perspective and the aggregator's expressive capacity.

In generic graphs, the numbers of neighboring nodes usually differ for different nodes in a graph, and there is no order information based on which we can order them to ensure the output is deterministic. These special characters of graph datasets require the aggregator should be a permutation-invariant function that can deal with variable-sized and unordered neighbors (**D1, D2**).

Permutation invariant study. Permutation invariance is an important property for aggregator since there is no natural order in most real graphs. The neighborhood aggregation scheme iteratively updates the representation of a node by aggregating representations of it neighbors. To mathematically formalize the above insight, the aggregation process can be generically written as follows:

$$\mathbf{s}_i^{(k-1)} = f_{ag}^{(k)}(\mathbf{h}_j^{(k-1)}, j \in \mathcal{N}_i), \tag{4.1}$$

where $f_{ag}^{(k)}$ is the predefined aggregation function (aggregator) in the k-th layer of a model.

The aggregator $f_{ag}^{(k)}$ can be seen as a function over the full multiset of node neighbors. Following [215], a multiset is a generalized concept of a set [232] that the same element can appear multiple times since different nodes can have identical feature vector. Recall that one of the desiderata is that the aggregator $f_{ag}^{(k)}$ should be a mutiset permutation invariant function. Following [232], a permutation invariant function on multiset can be defined as:

Definition 4.2.1. *A function f is permutation-invariant if*

$$f(\{\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_{|\mathcal{N}_i|}\}) = f(\left\{\mathbf{h}_{\pi_{(1)}}, \mathbf{h}_{\pi_{(2)}}, ..., \mathbf{h}_{\pi_{(|\mathcal{N}_i|)}}\right\}) \tag{4.2}$$

for any permutation π and $|\mathcal{N}_i|$ is the length of the sequence.

We will use $\Pi_{|\mathcal{N}_i|}$ to represent the mutiset of all permutations of the integers 1 to $|\mathcal{N}_i|$ and $\mathbf{h}_\pi$, $\pi \in \Pi_{|\mathcal{N}_i|}$, represents a reordering of the mutliset according to π. The following theorem in [232] shows the relation between set and permutation invariant function.

Theorem 4.2.1. *[232] A function operating on a multiset $\{\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_{(|\mathcal{N}_i|)}\}$ having elements from a countable universe, is a valid set function. It is invariant to the permutation of instances in the multiset, if it can be decomposed in the form $\rho(\sum_{\pi \in \Pi_{|\mathcal{N}_i|}} \phi(\mathbf{h}_\pi))$, for suitable transformations ϕ and ρ.*

The structure of permutation invariant function in Theorem 4.2.1 hints a general strategy for inference over mutiset. In other words, the key is to add up all representations and then apply nonlinear transformation.

Sum, mean, pooling aggregators and aggregators in GCN and GAT can be formulated as the format in 4.2.1, while CNN and LSTM can not.Thus our aggegator function can follow the sum, mean or pooling format. GCN and GAT add up all neighborhood neighbors with fixed weights or learnable weights, as shown in Eq. 4.3 and Eq. 4.4, respectively.

$$\mathbf{s}_i^{(k-1)} = f_{agg}^{(k)}(\mathbf{h}_j^{(k-1)}) = \sum_{j \in \mathcal{N}_i} \mathbf{h}_j^{(k-1)} / \sqrt{d_i d_j}. \tag{4.3}$$

where d_i and d_j are the node degrees of node v_i and node v_i respectively.

$$\mathbf{s}_i^{(k-1)} = f_{aga}^{(k)}(\mathbf{h}_j^{(k-1)}) = \sum_{j \in \mathcal{N}_i} \alpha_{ij} \mathbf{h}_j^{(k-1)}, \tag{4.4}$$

where α_{ij} is a learnable attention coefficient that indicates the importance of v_j to v_i. But convolution aggregator in LGCL is not permutation-invariant function, for the output of the aggregator will change if the inputs are reordered, and it can not deal with variable-sized data directly.

Discriminative power study. From the aggregator's expressive capacity, there are mainly three tasks: One is to learn the interactions between self node and its neighborhood (node-level distinguish); The second one is to learn the interactions between different dimensions of the node features, which will extract useful combinatory features automatically (feature-level distinguish). The final one is to discriminate graph structures (structure-level distinguish) (**D3, D4**).

Sum aggregator is an injective function in the structure level, while mean and pooling aggregators are not, which has been proved in [215]. Thus, we can design our aggregator function by extending the sum aggregator. Notice that this property may suit better for graph classification task where graph structure plays a key role. Adding up all neighbors' feature vectors may change the scale of the feature, which may not be good for node classification task. However, they all can not treat the neighborhood information differently in both node-level and feature-level.

The aggregation process in GCN and GAT, as shown in Eq. 4.3 and Eq. 4.4, can discriminate the neigborhood information in node-level, however all the features are treated equally within the feature vector $\mathbf{h}_j^{(k)}$, for each feature shares the same weight $(d_i d_j)^{-1/2}$ or α_{ij}. The convolution aggregator in LGCL allows for feature-level attention, but it is not an optimal choice to deal with variable size inputs and unordered graph datasets.

Algorithm 4.1 LA-GCN$_{\text{Mask}}$ (one iteration)

Input: $\mathcal{G}$= $(\mathcal{V}, \mathcal{E}, \mathbf{X})$ with N nodes;

Adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$;

Feature matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$;

Auxiliary model f;

Output: The learned representation for each node $\mathbf{h}_i^{(k)}$ (i = 1,2, ... N).

for each $v_i \in \mathcal{V}$ **do**

  for $j \in \mathcal{N}_i$ **do**

    calculate v'_j mask: $\mathbf{m}_j^{(k-1)} = f\ (\|\mathbf{h}_i^{(k-1)}, \mathbf{h}_j^{(k-1)})$

  end for

  mask aggregator: $\mathbf{s}_i^{(k-1)} = \sum_{j \in \mathcal{N}_i} \mathbf{h}_j^{(k-1)} * \mathbf{m}_j^{(k-1)}$

  feature transportation: $\mathbf{h}_i^{(k)} = \sigma(\mathbf{W}^{(k)}(\mathbf{h}_i^{(k-1)} + \mathbf{s}_i^{(k-1)}))$

end for

4.2.4 Mask Aggregator

Based on the theoretical study and analysis, we design our aggregator function by extending sum aggregation function. Besides, the expected aggregator could do feature-wise and node-wise modulation of the neighborhood information in the aggregation process, which naturally inspires us to filter the neighborhood information before aggregation and the mask aggregator are shown as following:

$$\mathbf{s}_i^{(k-1)} = f_{agm}^{(k)}(\mathbf{h}_j^{(k-1)}) = \sum_{j \in \mathcal{N}_i} \mathbf{h}_j^{(k-1)} * \mathbf{m}_j^{(k-1)}, \tag{4.5}$$

where $\mathbf{h}_j^{(k-1)} \in \mathbb{R}^{d_{k-1}}$, $\mathbf{m}_j^{(k-1)} \in \mathbb{R}^{d_{k-1}}$ is a specific mask for each neighbor, produced by the auxiliary model. Then we Hadamard product to multiply each neighbor and its corresponding mask before summation (**D3, D4**).

Theorem 4.2.2. *$f_{agm}^{(k)}$ is a permutation-invariant function acting on finite but arbitrary length sequences $\mathbf{h}_j^{(k-1)}, j \in \mathcal{N}_i$.*

Proof. Given $H = \left\{\mathbf{h}_1^{(k-1)}, \mathbf{h}_2^{(k-1)}, ..., \mathbf{h}_{(|\mathcal{N}_i|)}^{(k-1)}\right\}$, a finite multiset, and $\mathbf{h}_j^{(k-1)} \in \mathbb{R}^{d_{k-1}}$, our aggregator aims to fuse it into a fixed output $\mathbf{s}_i^{(k-1)} \in \mathbb{R}^{d_{k-1}}$ as follows:

$$\mathbf{s}_i^{(k-1)} = f_{agm}^{(k)}(\mathbf{h}_j^{(k-1)}) = \sum_{j \in \mathcal{N}_i} \mathbf{h}_j^{(k-1)} * \mathbf{m}_j^{(k-1)}, \tag{4.6}$$

where $\mathbf{m}_j^{(k-1)} \in \mathbb{R}^{d_{k-1}}$ is a specific mask for each neighbor, produced by the auxiliary model. We first feed the graph information to an auxiliary model to get a mask $\mathbf{m}_j^{(k-1)}$ for each node $\mathbf{h}_j^{(k-1)}$. For a trained auxiliary model, $\mathbf{m}_j^{(k-1)}$ is a specific and fixed mask (vector) for each neighbor's latent vector $\mathbf{h}_j^{(k-1)}$.

There exists a mapping function $\phi : \mathbb{R}^{d_{k-1}} \to \mathbb{R}^{d_{k-1}}$ that can formulate $\mathbf{h}_j^{(k-1)} * \mathbf{m}_j^{(k-1)}$ to $\phi(\mathbf{h}_j^{(k-1)})$, and Eq. 4.6 can be written as:

$$\mathbf{s}_i^{(k-1)} = f_{agm}^{(k)}(\mathbf{h}_j^{(k-1)}) = \sum_{j \in \mathcal{N}_i} \phi(\mathbf{h}_j^{(k-1)}), \tag{4.7}$$

and ρ can be seen as $\rho = 1$. Eq. 4.8 can be seen as a permutation of H, according to [232].

Next, we prove there exist an injective mapping function ϕ, so that $f_{agm}^{(k)}(\mathbf{h}_j^{(k-1)})$ is unique for each finite multiset H.

Since H is countable, each $\mathbf{h}_j^{(k-1)} \in$ H can be mapped to a unique element to prime numbers $p(H)$: $\mathbb{R}^M \to \mathbb{P}$ by a function $p(\mathbf{h}_j^{(k-1)})$. We can choose $\phi(\mathbf{h}_j^{(k-1)}) = -\log p(\mathbf{h}_j^{(k-1)})$. Therefore,

$$f_{agm}^{(k)}(\mathbf{h}_j^{(k-1)}) = \sum_{j \in \mathcal{N}_i} \phi(\mathbf{h}_j^{(k-1)}) = \log p(\mathbf{h}_j^{(k-1)}) \tag{4.8}$$

takes a unique value for each distinct H.

Besides, the dimension d_{k-1} of the latent space should be at least as large as the maximum number of input elements $|\mathcal{N}_i|$, which is both necessary and sufficient for continuous permutation-invariant functions [196].

For the universal approximation theorem [82], any continuous function can be approximated by a neural network, we can use mutlti-layer perceptrons (MLPs) to model and learn ϕ and $\rho = 1$. □

Besides the provement, we state the derivatives with regard to our aggregator. Assume parametrizations W_ϕ for ϕ, we have

$$\partial_{W_\phi}(\sum_{j\in\mathcal{N}_i}\phi(\mathbf{h}'^{(k-1)}_j)) = \sum_{j\in\mathcal{N}_i}\partial_{W_\phi}\phi(\mathbf{h}'_{j^{(k-1)}}),$$

this result shows the ordering is also irrelevant for the optimization process.

Theorem 4.2.2 shows that $f^{(k)}_{agm}$ of the multiset is a permutation-invariant function (**D2**). The learned mask can shows which features or neighbors are important, and filter the noisy information, which makes the aggregation results easier to explain and robust (**D5**).

4.2.5 Auxiliary Model

A natural follow-up question is how to get the mask $\mathbf{m}^{(k-1)}_j$. Under our framework, mask is learned from an auxiliary model and we hope the auxiliary model can 1) extract useful and high-level knowledge (e.g., focusing on important nodes and features) from neighborhood information to guide a better aggregation for the central node's representation learning; 2) deal with different sized and unordered input datasets without reorganization.

Motivated by this, we feed both central node and its neighbors into the auxiliary model. The auxiliary model can be an arbitrary neural network that has no requirement for size or order of the input datasets. The most commonly used RNN and CNN architecture can not be used as the auxilary model, because the input of CNN and RNN should be fix-sized and ordered, and they can not directly operate on graphs due to the irregular connectivity and lack of orderliness of nodes in a graph. On the contrary, the input of MLPs does not have such constrains, e.g., the order or number of the input data. Thus, we choose MLPs as the auxilarty model to learn the mask. Considering the trade-off between performance and efficiency, we apply an MLP with a single hidden layer.

Given node and its neighbors ($\{\ \mathbf{h}^{(k-1)}_i, \mathbf{h}^{(k-1)}_j, j\in\mathcal{N}_i\}$), we feed each node-neighbor pair to an auxiliary model, defined as following:

$$\begin{aligned}\mathbf{m}^{(k-1)}_j &= MLP^{(k)}(\|\mathbf{h}^{(k-1)}_i, \mathbf{h}^{(k-1)}_j) \\ &= \sigma(\mathbf{W}^{(k)}_m(\|\mathbf{h}^{(k-1)}_i, \mathbf{h}^{(k-1)}_j)),\end{aligned} \tag{4.9}$$

where σ is the activation function, e.g., sigmoid, RELU, $\mathbf{W}_m^{(k)} \in \mathbb{R}^{2d_{k-1} \times d_{k-1}}$ is the weight matrix and $\|$ denotes column-wise concatenation. The update rule for v_i is

$$\mathbf{h}_i^{(k)} = \sigma(\mathbf{W}^{(k)}(\mathbf{h}_i^{(k-1)} + \mathbf{s}_i^{(k-1)})), \tag{4.10}$$

where $\mathbf{W}^{(k)} \in \mathbb{R}^{d_k \times d_{k-1}}$ is the learnable weight matrix. After K iterations/layers, the final representation $\mathbf{h}_i^{(K)} \in \mathbb{R}^{d_K}$.

For multi-class node classification, $\mathbf{h}_i^{(K)}$ will be passed to a fully-connected layer with a $softmax$ activation function. The loss function is defined as the cross-entropy error over all labeled examples:

$$\mathcal{L} = -\sum_{l \in \mathcal{V}_l} \sum_{f=1}^{F} \mathbf{Y}_{lf} \ln \mathbf{h}_l^{(K)}, \tag{4.11}$$

where $\mathcal{V}_l$ is the set of node indices that have labels and d_K is the dimension of output features equaling to the number of classes. $\mathbf{Y}_{lf} \in \mathbb{R}^{|\mathcal{V}_l| \times F}$ is a label indicator matrix.

For graph classification, adding up all $\mathbf{h}_i^{(K)}$ or more sophisticated graph-level pooling can be applied to get the entire graph's representation.

4.2.6 Computational Complexity

A key part in our method is the auxiliary model, and it is a shared model by all nodes in a graph. So, the computation of the mask can be parallelized across all nodes, which is highly efficient. The computational complexity of Eq. (4.10) is $\mathcal{O}(\mid \mathcal{E} \mid \times d_k \times d_{k-1} + \mid \mathcal{E} \mid \times 2d_{k-1} \times d_{k-1})$ and is in par with GCN ($\mathcal{O}(\mid \mathcal{E} \mid \times d_k \times d_{k-1})$). As for the memory requirement, it grows linearly in the size of the dataset and we perform mini-batch training to deal with this issue.

4.2.7 Differences with Existing GNNs

We compare our model with prominent GCN based models and we study all these model from three aspects:

- **Aggregator** Sum [215] and mean [72] are two most commonly seen aggregators. The aggregator in GCN [103] can be seen as a weighted mean aggregator (mean_w), and the weight is $(d_i d_j)^{-1/2}$, where d_i, d_j are the node degree of central node v_i and

neighbor v_j. In GAT [191], the aggregator is a learnable weighted summation (sum_{lw}). Convolutional operation (*Conv*) is used to aggregate the neighborhood information in LGCL [58] and NFC-GCN in Sec. 3.1. Our aggregator function can be seen as an extention of sum_{lw}, which applies learnable masks to filter the neighborhood information before summation. We summarize the relationship between desiderata and mentioned aggregator in Table 4.1. Our aggregator satisfies all desiderata, enabling a leap in model capacity. Furthermore, analyzing the learned mask may lead to benefits in interpretability.

- **Auxiliary model**. GCN, GraphSAGE [72] and GIN [215] do not use any auxiliary model to guide the aggregation process, and sum or mean the neighborhood directly. while a shared convolutional layer and a shared single-layer feed forward neural network are used in Sec. 3.1, LGCL and GAT respectively. Considering the limitation of CNN and RNN whose input data should be ordered and fixed-size, we use a shared single-layer feed forward neural network. Both our model and MaskedGCN aim to learn the mask. While, there are two main differences between our auxiliary model and MaskedGCN. First, MaskedGCN actually does not use any auxiliary model to learn the mask, and the mask is a learnable vector that contains the learnable parameters directly. There is no parameter sharing among nodes, and model's parameters grow with the number of nodes in a graph, which is computationally inefficient. While, our model applies a shared MLP among all nodes to learn the mask and it is highly efficient. Second, our model applies MLP to model the node-neighbor (v_i-v_j) pairs directly to learn the mask. While the mask vector in MaskedGCN is learned independent of the central node v_i, and depends on the consistencies of each feature between v_j and v'_j neighbors.

- **Input and output of the auxiliary model**. For GAT, the input is node-neighbor pairs (input), and the auxiliary model learns from them to get coefficients (output) between nodes, which allows the aggregator to focus on most relevant nodes. The aggregator adds up each neighbor corresponding to the learned weight to get the aggregation

Table 4.2 Overview of datasets for graph classification.

Datasets	Graphs	Classes	Avg. nodes
MUTAG	188	2	18
PROTEINS	1,113	2	39
PTC	344	2	26

output. However, GAT only learns node-level attention. LGCL uses the reorganized neighbor's embedding (input) that selects the d-largest values for each feature from neighbors to calculate the convolutional filter's weights (output). NFC-GCN selects and orders a fixed-size neighbors as input and learns the convolutional filter's weights (output). LGCL and NFC-GCN allow for feature-level attention, but they can not deal with variable size inputs (the number of adjacent nodes usually varies for different nodes in a graph), due to the limitation of convolution operation. While we concatenate the central and neighbor before feed in the auxiliary model, which can be viewed as a simple form of a " skip connection " between different search depths and get the learned mask for each given node's neighbor.

4.3 Experiments

We perform evaluation on node classification and graph classification, and study our model's interpretability and robustness.

4.3.1 Datasets

We conduct node classification on three citation graphs (Cora, Citeseer and PubMed) and one social network (Reddit), which have been widely used in [103, 36, 216, 72, 191, 58, 1, 242] and graph classification on 3 bioinformatics datasets summarized in Table 4.2.

- **Node classification.** We use the citation graphs as mentioned in Sec. 3.3.1. Reddit is a large online discussion forum where users post and comment on content in different topical communities. The node label is the community, or "subreddit", that a post

belongs to. The link means the same user comments on both posts. Hamilton et al. [72] concatenates the average embedding of the post title, the average embedding of all the post's comments, the post's score, and the number of comments made on the post as node features. Reddit contains 41 discrete labels, 232,965 nodes, 11,606,919 links, and node features, a 602-dimensional vector for each node, are learned from user's text information: post title, post comments, post score and the number of comments. Following Hamilton et al. [72], we divide the train/validation/test as 152K, 23K and 55K respectively.

- **Graph classification.** We use 3 bioinformatics datasets [215]. MUTAG is a dataset of 188 mutagenic aromatic and heteroaromatic nitro compounds with 7 discrete labels. PROTEINS is a dataset where nodes are secondary structure elements (SSEs) and there is an edge between two nodes if they are neighbors in the amino-acid sequence or in 3D space. It has 3 discrete labels, representing helix, sheet or turn. PTC is a dataset of 344 chemical compounds that reports the carcinogenicity for male and female rats and it has 19 discrete labels.

4.3.2 Baselines

Node classification. We compare against 6 strong baselines: GCN [103], GAT [191], FastGCN [36], GraphSAGE-mean [72], LGCL [58] and MixHop [1] using the publicly released implementations. We split the train/validation/test as in 3.3.1.

Graph Classification. Following [215], we compare our model with the following baselines:WL subtree [166], DCNN [11], PATCHYSAN [143], DGCNN [239], AWL [91] and GIN with its variants.

4.3.3 Hyperparameters Setting

Node classification. In our model, we first utilize one GCN layer to reduce the dimension of the node feature to 64-dimension for Cora, PubMed and 128-dimension for Citeseer and Reddit. Then we apply a one-layer neural network as the auxiliary model to learn masks

Table 4.3 Node classification accuracy (%) (mean ± 95% confidence interval over 100 runs) . The best results are in **bold** and the second best ones are underlined.

Methods	Cora	Citeseer	PubMed	Reddit
GCN	88.1 ± 0.235	77.8 ± 0.216	86.8 ± 0.176	93.0 ± 0.261
GAT	80.4 ± 0.255	75.7 ± 0.431	85.0 ± 0.039	–
LGCL	86.9 ± 0.216	77.5 ± 0.392	84.1 ± 0.118	–
FastGCN	85.0 ± 0.470	77.6 ± 0.124	88.0 ± 0.627	93.7 ± 0.365
GraphSAGE	82.2 ± 0.135	71.4 ± 0.165	87.1 ± 0.921	94.6 ± 0.217
MixHop	88.3 ± 0.446	–	85.6 ± 0.329	–
NFC-GCN	88.7 ± 0.255	**79.4 ± 0.431**	**89.7 ± 0.137**	94.9 ± 0.517
Ours	**89.1 ± 0.321**	78.7 ± 0.352	89.1 ± 0.252	**95.1 ± 0.264**

for neighbors, whose input dimension is 128×64 (Cora, PubMed) and 256×128 (Citeseer and Reddit). Hyperparameters are optimized with the validation set [36]. Throughout the experiments, we use the Adam optimizer [102] with learning rate 0.05 for Cora and PubMed, 0.002 for Citeseer, and 0.01 for Reddit. We fix the dropout rate to 0.5 for the hidden layers' inputs and add an L2 regularization of 0.0001. We employ the early stopping strategy based on the validation accuracy and train 200 epochs at most. For Reddit, we use the mini-batch training and the batch size (512) is set to be the same as FastGCN and GraphSAGE.

For a fair comparison, we also use the hidden layer size of 64 units for GCN on Cora, PubMed and 128 for Citeseer, which ensures the architecture is the same with ours model (except the auxiliary model part). We use the same architecture as in the original papers for GAT, LGCL, FastGCN, GraphSAGEMixHop and NFC-GCN. We report results over 100 runs with random weight matrix initialization.

Graph classification. We report the mean accuracy of 100 runs with random weight initialization for all of our experimental results. We use the following hyperparameters for MUTAG, PTC and PROTEINS: 0.005 (learning rate), 16 (the number of hidden units), 0.5 (dropout ratio), 32 (batch size). We replace the sum aggregator in GIN with our learnable aggregator and MLPs with two layers are applied after aggregation. Batch normalization [102] is applied on each hidden layer.

Table 4.4 Graph classification accuracy (%) (mean ± 95% confidence interval over 100 runs) . The best results are in **bold** and the second best ones are underlined.

	Methods	MUTAG	PROTEIN	PTC
Baselines	WL subtree	90.4 ± 2.93	75.0 ± 1.76	59.9 ± 2.66
	DCNN	67.0	61.3	56.6
	PATCHYSAN	**92.6 ± 2.25**	75.9 ± 1.94	60.0 ± 3.29
	DGCNN	85.8	75.5	58.6
	AWL	87.9 ± 4.77	–	–
GNN variants	GIN	89.4 ± 2.55	76.2	64.6 ± 3.98
	Sum-1-Layer	90.0 ± 4.83	76.2 ± 2.03	63.1 ± 4.13
	Mean-MLP	83.5 ± 4.25	75.5 ± 2.03	66.6 ± 4.76
	Mean-1-Layer	85.6 ± 3.60	76.0 ± 1.79	64.2 ± 4.19
	Max-MLP	84.0 ± 4.39	76.0 ± 1.83	64.6 ± 4.73
	Max-1-Layer	85.1 ± 4.57	75.9 ± 2.77	63.9 ± 5.03
	LA-GCN$_{\text{Mask}}$ (Ours)	90.0 ± 3.06	**80.5 ± 2.03**	**72.2 ± 4.88**
	Improvement	-	4.30	5.60

Table 4.5 Node classification with different label size (%). The best results are in **bold** and the second best ones are underlined.

Datasets	Methods	1%	2%	3%	4%	5%	10%	20%	30%	40%	50%	Average
Cora	GCN	58.41	71.70	75.83	79.27	82.28	**85.50**	85.57	**86.93**	87.00	86.27	79.87
	GAT	45.31	58.79	68.12	71.23	77.49	85.20	85.90	86.70	87.10	86.20	74.66
	LGCL	60.58	71.25	75.55	79.28	**82.53**	84.85	**86.03**	86.58	86.85	87.13	80.06
	Ours	**63.50**	**73.61**	**76.70**	**79.49**	81.11	84.34	85.58	86.58	**87.68**	**87.72**	**80.60**
Citeseer	GCN	42.76	69.29	71.66	**72.50**	**73.32**	**76.90**	77.77	77.93	77.83	78.17	70.93
	GAT	47.78	63.57	54.38	50.48	72.10	75.40	74.60	75.20	77.00	77.02	64.95
	LGCL	**57.80**	66.92	**72.32**	71.28	73.10	76.34	76.38	76.86	77.07	77.02	72.51
	Ours	57.35	**69.52**	71.02	72.03	**72.16**	76.48	**78.49**	**78.12**	**79.29**	**79.35**	**73.26**
PubMed -	GCN	79.92	80.46	79.18	79.28	79.62	82.47	84.30	83.40	84.70	85.07	81.48
	GAT	78.56	79.48	78.02	78.62	78.64	81.60	83.00	83.20	83.20	83.20	80.48
	LGCL	**81.95**	**82.70**	**83.10**	82.93	81.30	82.50	85.37	84.60	85.46	85.74	83.32
	Ours	80.00	82.44	81.35	**83.13**	**82.67**	**85.50**	**86.70**	**87.50**	**87.50**	**87.30**	**84.09**

4.3.4 Performance for Node Classification

Results for node classification and graph classification are summarized in Table 4.3 and Table 4.4. We observe that LA-GCN$_{\text{Mask}}$ outperforms all the mentioned methods across all datasets except MUTAG.

For node classification, GCN outperforms GAT, which is consistent with the results reported in [216, 1]. FastGCN and GraphSAGE focus on improving the training efficiency so they have slightly worse results than GCN. LGCL reorganizes the original embedding in the process of constructing feature maps and it does not perform well particularly on PubMed. MixHop utilizes different hop neighbors information and gets the second best performance on Cora, but does not perform well on Citeseer and PubMed. One possible reason is that it does not filter the neighborhood information, which may aggregate some noisy information from higher-order neighbors. NFC-GCN allows for feature-level attention of the local neighborhood information and gets slightly better results. One possible reason is that NFC-GCN only selects most related neighbors, and has many convolutional kernels, which could be more expressive than LA-GCN.

4.3.5 Performance for Graph Classification

For graph classification, Table 4.4 compares LA-GCN$_{\text{Mask}}$ with GIN, other GNN variants, as well as other strong baselines. In general, GNN variants perform better than the mentioned baselines, and the main reason is that they can not combine node features, which might limit the models' capacity. GNNs with sum aggregator tend to fit the training sets better than mean and max-pooling aggregators. Further, we can see that replacing the sum aggregator in GIN can significantly improve the accuracy on PROTEIN and PTC datasets by 4.5% and 5.6%, excluding MUTAG. One possible reason for the poor performance on MUTAG is that our model may not be fully trained due to the small training sample size.

4.3.6 Parameter Sensitivity

We also compare our method with closely related methods, GCN (mean$_w$ aggregator), GAT (sum$_{lw}$ aggregator) and LGCL (*Conv* aggregator), in two scenarios: small training size (1%, 2%,..., 5% for Cora and Citeseer, 0.5%, 0.6%, 0.7%, 0.8%, 0.9% for PubMed [1]) and large training size (10%, 20%,..., 50%) for Cora, Citeseer and PubMed. Results are summarized in

[1]Compared with Cora and Citeseer, PubMed has more nodes. So, we choose the training size with smaller percentages.

Table 4.5. Note that Reddit is too large for GAT, so we only report results on three citation graphs and accuracy of 100 runs with random weight initialization for all of them.

Table 4.5 shows node classification results with different training sample sizes. The 95% confidence interval belongs to (0-1.0) and we did not show this in Table 4.5 for a better view. On the whole, our model has achieved competitive performance in small training sample size and got better performance, especially with more training data. The main reason is that our model has more parameters than GCN, and we briefly summarized the number of parameters of GCN, GAT and LA-GCN. GCN's parameter is less than ours on Cora, Citeseer and PubMed 8.16%, 6.46% and 20.0%, respectively. Compared with GCN, our model need more training samples. For a more intuitive comparison, we average these results for each method under different training size. LGCL and LA-GCN$_{\text{Mask}}$ outperform GCN and GAT, which indicates that being discriminative to feature-level is crucial for node classification.

4.3.7 Effectiveness of Mask Aggregator

To show the effectiveness of our aggregator, we compare our aggregator with three fundamental aggregators: mean, sum and maxpooling[2]. Results are summarized in Table 4.7.

Table 4.6 Node structure and feature statistics. (H.Nd.: Highest Node degree, L.Nd.: Lowest Node degree, M.Nd.: Median Node degree, and A.Nd: Average node degree. Fea.De. means feature density).

Dataset	H.Nd.	L.Nd.	M. Nd.	A.Nd	Fea.De.
Cora	168	1	4	4.9	1.26%
Citeseer	99	1	3	3.7	0.84%
PubMed	171	1	3	5.5	1.00%

Table 4.7 shows that our aggregator works better on Cora and PubMed than other aggregators, but not on Citeseer. The main reason is that Citeseer is more sparse in both graph structure and node feature, as shown in Table 4.6. The median of the neighbors'

[2]We use the same model architecture, besides the aggregator, and we name them as: GCN$_{mean}$, GCN$_{sum}$ and GCN$_{pooling}$.

Table 4.7 Different aggregators for node classification (%).

Dataset	Cora	Citeseer	PubMed
GCN_{mean}	87.7 ± 0.216	77.7 ± 0.276	86.0 ± 0.133
GCN_{sum}	85.5 ± 0.540	77.1 ± 0.552	85.2 ± 0.535
$GCN_{pooling}$	84.7 ± 0.324	$\mathbf{79.1 \pm 0.579}$	86.2 ± 0.324
Ours	$\mathbf{89.1 \pm 0.175}$	78.7 ± 0.260	$\mathbf{89.1 \pm 0.216}$

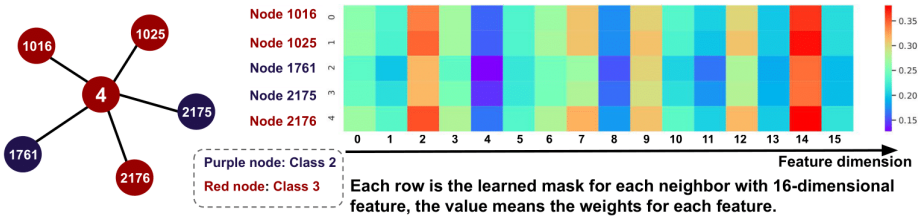


Fig. 4.3 Visualization of the learned mask. The proposed aggregator can focus on important neighborhood information (e.g. the neighbors from the same class, or some highly relevant features) with the learned mask. The values showed in the heat map are the real values of the weights.

number is 3 and the feature density is 0.84% (3703 is divided by 18, the average number of "1" in a feature vector). So, max-pooling may be the best way that can collect most information from the neigborhood, which benefits the later feature transformation stage. But the result of pooling aggregator may not be very stable, and the standard deviation (3.26%) is almost ten times higher than other aggregators on Cora. As for mean and sum aggregator, mean aggregator performs better both in accuracy and stability in general. Adding up all neighbors' feature vectors may change the scale of the feature, which may not be good for node classification task.

4.3.8 Interpretability Study

Sum, mean, pooling and *Conv* aggregators mix or reorganize neighborhood information, which makes it difficult to interpret the learned representation because we can not distinguish which node and feature have a salient influence on the prediction result. While our aggregator provides a learned mask for each neighbor, which provide a qualitative and quantitative understanding of the relationship between input nodes and the prediction. In this subsection,

we aim to answer the following **Q**uestions: **(1)** what we expected, **(2)** what we learned and **(3)** what we concluded.

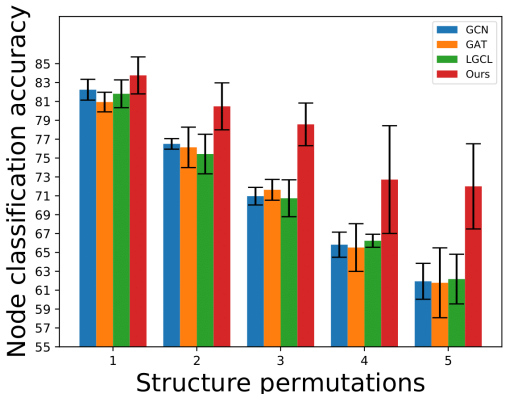
For **Q1**, the expectation intuitively is that the learned mask should assign more weights to important neighbors and features.

In order to answer **Q2**, we visualize the learned masks for a representative node: Node 4 in Cora with neighbors from different classes, as shown in Fig. 4.3. Central node 4 and its neighbors 1016, 1025 and 2176 belong to the same class, while neighbors 1761, 2175 belong to another class (class 2). From Fig. 4.3, we see that neighbors (1016, 1025, 2176) from the same class are assigned more weights (the values in the learned mask) than the other two neighbors (1761, 2175) on the whole. Besides, the mask gives high importance scores to some specific feature dimensions. We also analyze how GCN and GAT aggregate node 4's neighbors. GCN assigns weights - 0.2, 0.16, 0.16, 0.17, 0.2 to nodes 1016, 1025, 2176, 1761, 2175 respectively, depending on node 4 and its neighbors' node degree. The neighbors are treated differently in node-level, but it is not as we expected. It is reasonable to expect that node 1025 and 2176 (from the same class with central node) should be given higher scores than node 1761 and 2175. For GAT, the learned attention weights are all around 0.17, and the neighbors are not treated significantly differently.

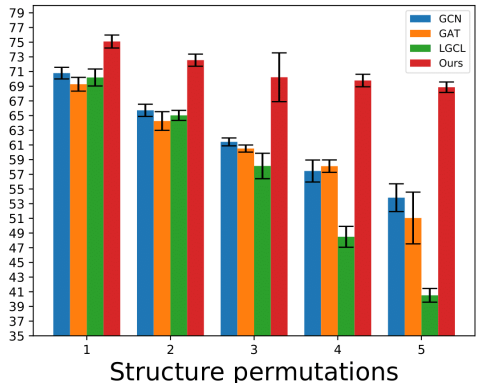
This indicates that the auxiliary model learns the expected rules (focusing on the important neighbors and features), which is used to assist our aggregator to jointly consider node-level as well as feature-level modulation of neighborhood information in the aggregation process (**Q3**). However, all features in one feature vector share the same weights in both GCN and GAT.

4.3.9 Robustness Study

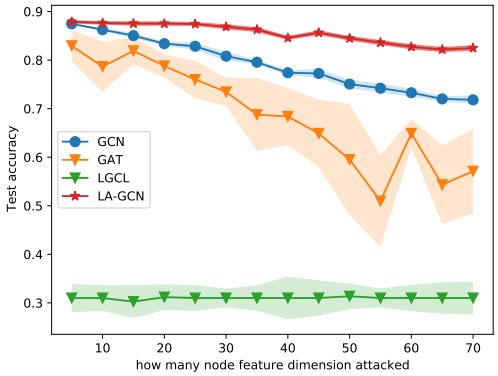
Because real-world graphs are noisy, an essential criterion is that the model should be robust. As shown in [253], permutations to both graph structures and node features are harmful. To study the robustness of LA-GCN$_{\text{Mask}}$, we test our model on both structure noisy graphs, i.e., changes to adjacency matrix, and node feature noisy graphs, i.e., changes to node feature matrix.



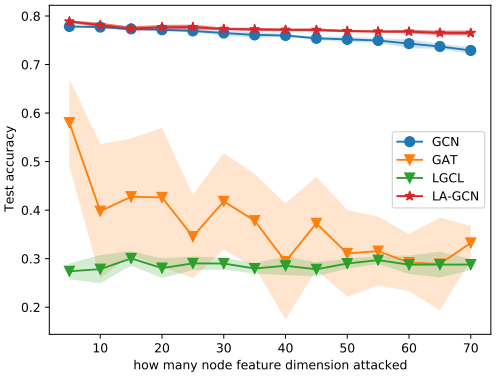
(a) Cora with structure noise



(b) Citeseer with structure noise



(c) Cora with feature noise



(d) Citeseer with feature noise

Fig. 4.4 Robustness studies: (a) and (b) show the node classification accuracy on structure noisy graphs, and (c) and (d) show the node classification accuracy on node feature noisy graphs.

We follow [41] to utilize the simplest attack methods. Given a target node, we randomly delete or add edges to the graph. For structure attack, the budget for each node is from one to five, which means that we are allowed to randomly add or delete one to five neighbors for each node. Following [255], per-node changes to the node attributes are at most 5% of the node feature dimension. The node feature vector in Cora and Citeseer only contains 0 or 1, so we randomly flip the features for feature attack. We compare our model with GCN, GAT and LGCL on both structure and node feature noisy graphs. Considering the unstable

problem caused by the noisy data for these models, we report the average of top 10 results over 40 runs for each method, as shown in Fig 4.4.

Figure 6.6a and Fig. 4.4b shows that the performance gets worse with the attack budget increasing. Our model gets the best performance, especially with more structures changed. When the structure permutation is 5, the second best can only achieve 62% and 54 % classification accuracy on Cora and Citeseer respectively, while ours are 72% and 68%. In this case, the feature vector of the central node is still well preserved and our aggregator can effectively identify those features good for the classification of the central node from noisy neighborhood information.

For node feature noisy graphs, as shown in Figs. 4.4c and 4.4d, GAT and LGCL degrades significantly and our method shows strong robustness. Compared with GCN, the improvement is not as significant as in structure attack experiments. In this scenario, the central node's feature is also polluted in some extend, which may mislead the learned mask in the neighborhood aggregation process.

4.4 Summary

Considering most real-world graphs with no regular connectivity and order, we further extends NFC-GCN by lifting constraints on the input data format and proposed a new model to solve the first research question in this chapter. We unified current aggregators in a framework: LA-GCN, with an auxiliary model to guide the neighborhood aggregation process. We carefully designed the auxiliary model under this framework and proposed a new aggregator: mask aggregator that learns a specific mask for each neighbor, which allows end-to-end training and both node-level and feature-level attention for neighborhood information. LA-GCN$_{\text{Mask}}$ provides a variety of benefits, from an easy implementation with a much better performance, to interpretability, to robustness in noisy graphs. We evaluated LA-GCN$_{\text{Mask}}$ against six state-of-the-art methods on variable type and size graphs for node classification and six strong baselines on graph classification. Experimental results showed the superior performance of LA-GCN$_{\text{Mask}}$ over other methods on the whole, particularly a

remarkable improvement on noisy graphs. Furthermore, analyzing the learned mask provided a straightforward interface for make sense out of prediction and quantified understanding of the relationship between input nodes and prediction. In addition, the proposed mask aggregator can be integrated with other GCN variants such as FastGCN [36], jumping knowledge networks [216], GMWW [154] and MixHop [1].

Chapter 5

Hop-Hop Relation-aware Graph Neural Network

5.1 Introduction

Real-world objects and interactions are often heterogeneous (e.g., movies, actors and directors in knowledge graph (KGs), authors, papers, venues in a publication network [167]). To capture and exploit such node and link heterogeneity, heterogeneous graphs have been proposed and widely used in many real-world graph mining scenarios [218]. For node representation learning, only the first-order neighborhood information may be not enough, which is caused by the sparsity or heterophily [62, 61, 214]. To learn a better representation, it would be better to leverage multi-scale neighborhood information, which enables the model to leverage more information or to explore the same type of nodes at various distances. In Chapter 3 and Chapter 4, we mainly focus on how to design new GNN models for feature-level and node-level attentions of first-order neighborhood information. In this chapter, we mainly focus on how to design the GNN model to leverage different hops of neighbors, meanwhile incorporate hop-level attention of local neighborhood information.

The majority of current GNNs focus on homogeneous graphs, and the aggregator aggregates a central node's one-hop neighbors [103, 191], neighbors sampled from fixed-length random walks [72] or multi-scale neighbors [1, 120, 11] in the neighborhood aggregation

step. In recent years, there are some attempts to apply GNNs to heterogeneous graphs [161, 206], in which different types of nodes are connected under unique relations [218]. Most existing GNNs-HE aggregate the neighbors from manually designed or automatically learned meta-paths [206, 164]. However, it remains unclear which hops or types of neighbors is crucial for the central node's representation learning? GNNs-HO only aggregate the neighbors directly and fail to exploit the relationship between central node with different hops of neighbors. GNNs-HE only aggregate two end nodes in the manually defined meta-path and discard all intermediate nodes along the meta-path, which could lead to information loss and the real meaningful relations for each node can not be explored.

Based on the above analysis, the key is to design a new neighborhood aggregation framework that can automatically learn the relationship between central node with different hops or types of neighbors. Meanwhile, we also need to consider two key differences of GNNs-HO and GNNs-HE in the neighborhood aggregation process. 1) How to define the receptive field, which is determined by the powers of adjacency matrix and meta-paths in GNNs-HO and GNNs-HE respectively. 2) How to aggregate the neighbors. Most GNNs-HO aggregate all neighbors in the receptive field and GNNs-HE only aggregate two end nodes in a meta-path. In summary, the new aggregation process should satisfy the following **Desirables**:

- **D1**: **A personalized (learnable) receptive field**. GNNs-HO and GNNs-HE use different ways to define their receptive fields. A common issue is that the receptive fields are more or less hand-designed based on the powers of adjacency matrix or meta-paths. Besides, considering the connectivity for each node varies greatly in a graph [191, 139], previous mechanism may not be suitable for each node and limit GNNs to discover meaningful receptive fields from a graph. Therefore, we need an automated way of learning the receptive field for each node.

- **D2**: **Hop-aware projection**. Different hops or types of neighbors have different traits and their embeddings should be mapped in different feature spaces [161]. Different types of neighbors along a meta-path in heterogeneous graph can be treated as different

Table 5.1 Comparisons of other GNNs and our model: HHR-GNN. Outline of related work in term of fulfilled ($\surd$) and missing ($\times$) desirable characteristics.

Desirables	GCN	GAT	GraphSAGE	GinePath	HAN	GTN	GTN
D1	$\times$	$\times$	$\times$	$\surd$	$\times$	$\times$	$\surd$
D2	$\surd$	$\surd$	$\times$	$\surd$	$\times$	$\surd$	$\surd$
D3	$\times$	$\times$	$\times$	$\times$	$\surd$	$\surd$	$\surd$
D4	$\surd$	$\times$	$\surd$	$\times$	$\surd$	$\times$	$\surd$
D5	$\times$	$\surd$	$\times$	$\times$	$\times$	$\surd$	$\surd$

hops of neighbors in homogeneous graphs. Hop-aware projection is an important requirement before the aggregation for the two types of graphs.

- **D3**: **Hop-aware aggregation**. In GNNs-HO, different hops or types of neighbors show different importance for central node's representation learning. For example, the directly linked (one-hop) neighbors have a closer relationship with the central node than indirectly linked (higher-order) neighbors in homogeneous graph, while it could be the opposite for heterogeneous graphs with a high probability, e.g., two papers can be connected by the meta-path Paper-Author-Paper (*PAP*) in a citation graph, and the two-hop neighbors (P-type nodes) have a closer relationship (papers written by the same author) than one-hop neighbors (A-type nodes) with the central node (P-type nodes). Therefore, the aggregation process should be smarter and GNNs should be able to model different relationships between a central node and its different hops or types of neighbors during aggregation.

- **D4 & D5: Efficiency and interpretability**. For practical applications where efficiency and interpretability are needed [222], the model should be able to explain which hops or types of neighbors play important roles for the central node's representation learning. Moreover, the model should deal with the complex neighborhood in an efficient way, especially for large-sized and heterogeneous graphs [47, 206].

To this end, we propose a general approach that can satisfy the above desirables for both homogeneous and heterogeneous graphs, Hop-Hop Relation-aware Graph Neural Network (HHR-GNN), which mainly contains three modules: GNN, Knowledge Graph Embedding

(KGE) and aggregation. In GNN module, we first utilize different mapping matrices (**D2**) for different hops (types) of neighbors to learn hop-specific representations. [1] In order to model the relationship between the central node and its different hops or types of neighbors, we introduce a complementary module: KGE which can capture the interaction between *head* and *tail* entities by learning their relation-score in knowledge graphs (KG) [204]. We feed the embeddings learned from GNNs to the KGE module to model the relationship between a central node and its different hops or types of neighbors. In this chapter, we utilize four representative KGE methods: RESCAL [142], DistMult [217], MLP [44], NTN [171] to learn the relation-scores [2] between central node and its different hops' representations. This will enable the central node to softly aggregate neighborhood information at different hops (**D1**). The relation-scores are learned from the low-dimensional and fix-sized embeddings [3], so the learning process can be done efficiently (**D4**). Finally, we apply six commonly used aggregators: sum, mean, max pooling, concatenation, LSTM, GRU [72] to aggregate the central node and its different hops' embeddings weighted by their corresponding relation-scores (**D3**) to get the central node's final embedding, which allows our model to automatically aggregate latent information from neighbors at various distances and types. Analyzing the learned relation-scores can show which hops or types of neighbors are important and this benefits interpretability (**D5**). Finally, we compare our proposed aggregator and the mentioned aggregators in Table 5.1 to show our novelty. Our proposed model, HHR-GNN, satisfies all desiderata, enabling a leap in model capacity.

We evaluate HHR-GNN on two popular homogeneous graphs and three heterogeneous graphs for node classification. Our results confirm that our model performs well on both types of graphs. In addition, we visualize the learned relation-scores to show that our model can automatically identify which hops or types neighborhood information is important for the central node's representation learning, which provides an interpretable explanation for the prediction. We further compare the efficiency of our model with others and our model outperforms state-of-the-art GNNs-HO and GNNs-HE, with up to 13K faster in term of time

[1] p-hop representation contains all neighbors' information at p-hop ($p = 0, 1, 2...$).
[2] For a given node, we assume neighbors in the same hop have the same relation-score.
[3] The size is decided by the number of hops, not the number of neighbors.

cost. Finally, we explore the application of different KGE models in HHR-GNN and find different problems favor different KGE models depending on training dataset size.

In summary, the key contributions of this chapter are:

1. We propose Hop-Hop Relation-aware Graph Neural Networks (HHR-GNN), a new framework of GNNs, that can model the relationship between central nodes with their different hops or types of neighbors.

2. HHR-GNN is suitable for both homogeneous and heterogeneous graphs representation learning, meanwhile it solves some issues of current models, such as fix-sized receptive fields, no hop distinction or information loss in the neighborhood aggregation process.

3. We comprehensively evaluate the superiority of our model on both homogeneous and heterogeneous graphs for the popular node classification tasks. Besides, HHR-GNN can both ensure the accuracy and efficiency (up to 13K faster in terms of time cost) while providing an interpretable explanation for the prediction.

5.2 Methodology

This section presents our proposed HHR-GNN as shown in Figure 5.1. Borrowing ideas from knowledge graph embedding methods, HHR-GNN is a general neighborhood aggregation framework that can manage the relationship of central node and its different hops or types of neighbors during the neighborhood aggregation. After the notation and problem definition, we first start from theoretical study of homogeneous and heterogeneous graphs and their neighborhood aggregation in GNNs-HO and GNNs-HE. This enables us to derive a general framework that is suitable for both homogeneous and heterogeneous graphs representation learning and satisfies all the desirables mentioned above. Finally, we analyze our model's complexity and differences with existing GNNs.

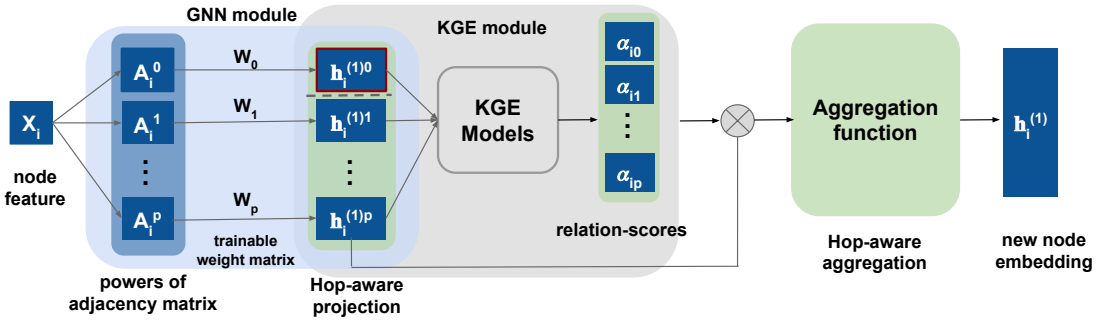


Fig. 5.1 HHR-GNN architecture (the first layer): HHR-GNN first calculates its representations at different hops, e.g., $\mathbf{h}_i^{(1)_p}$ is p-hop representation that aggregates neighbors in the p-hop. $\mathbf{A}_i^r$ is the i-th row of r-th power of adjacency matrix. Then the central node's representation $\mathbf{h}_i^{(1)_0}$ and its representations at different hops $(\mathbf{h}_i^{(1)_1}, \mathbf{h}_i^{(1)_2}, ..., \mathbf{h}_i^{(1)_p})$ will be fed to a NTN model to learn the relation-scores. Finally, we concatenate each node's embedding with its representations at different hops weighted by their corresponding relation-scores to get the new embedding.

5.2.1 Problem Definition

The problem we consider in this chapter is semi-supervised node classification task on graphs. A graph with N nodes can be represented as $\mathcal{G}$= $(\mathcal{V}, \mathcal{E}, \mathbf{X})$, where node $v_i \in \mathcal{V}$, edges $(v_i, v_j) \in \mathcal{E}$ $(i, j = 1, ..., N)$, and a feature matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$ containing N D-dimensional feature vectors. The hidden representation learned by the k-th layer of a GNN model is denoted by $\mathbf{H}^{(k)}$ and we initialize $\mathbf{H}^{(0)} = \mathbf{X}$. The goal of GNNs is to learn meaningful and low-dimensional node embeddings, which can be used for down-stream tasks, such as node classification, link prediction or other tasks.

This chapter focuses on the semi-supervised node classification task. $\mathcal{V}_l$ is the set of nodes with labels $\mathbf{Y}_{lf} \in \mathbb{R}^{|\mathcal{V}_l| \times F}$, a label indicator matrix, F is the number of classes, the rest of the nodes $\mathcal{V}_u$ are unlabeled. We aim to predict the labels for a subset of unlabeled nodes.

5.2.2 Theoretical Studies

In this subsection, we mainly reveal the relationship between neighborhood aggregations in GNNs-HO and GNNs-HE.

Homogeneous and Heterogeneous Graphs

This section first introduces how to represent the two types of graphs and the relationship between **hops** in homogeneous graph and **types** in heterogeneous graph.

As defined in Definition. 2.1.3, heterogeneous graph is associated with a node type mapping function f_v: $\mathcal{V} \rightarrow \mathcal{T}_v$ and a link type mapping function f_e: $\mathcal{E} \rightarrow \mathcal{T}_e$, where $|\mathcal{T}_v| + |\mathcal{T}_e| > 1$. If both $\mathcal{T}_v$ =1 and $\mathcal{T}_e$ =1, it is a homogeneous graph with the same type of nodes and edges.

The heterogeneous graph can be represented by a set of adjacency matrices $\{\mathbf{A}_r\}_{r=1}^{R}$ (R=$|\mathcal{T}_e|$), and $\mathbf{A}_r \in \mathbb{R}^{N\times N}$ is an adjacency matrix where $\mathbf{A}_r[i,j]$ is non-zero when there is a r-th type edge from v_j to v_i. In homogeneous graphs, the adjacency matrix is simplified to $\mathbf{A} \in \mathbb{R}^{N\times N}$ (R=1).

In heterogeneous graph, two nodes can be connected via different semantic paths, which are called meta-paths. Meta-path defines a composite relation $R = r_1 \circ r_2 ... \circ r_p$ between node v_1 and v_{p+1}. Given the composite relation R, the adjacency matrix of the meta-path can be obtained by multiplications of adjacency matrices as:

$$\mathbf{A}_R = \mathbf{A}_{r_1}\mathbf{A}_{r_2}...\mathbf{A}_{r_p}. \tag{5.1}$$

For example, two Authors can be connected by the meta-path Author-Paper-Author (APA) via the meta path $A \xrightarrow{AP} P \xrightarrow{PA} A$ (a two-step walk), and the co-author (AA) graph can be obtained by the multiplication of $\mathbf{A}_{AP}$ and $\mathbf{A}_{PA}$. In homogeneous graph, Eq. 5.1 can be written as

$$\mathbf{A}_R = \mathbf{A}^p, \tag{5.2}$$

because $\mathbf{A}_{r_1} = \mathbf{A}_{r_2} ... = \mathbf{A}_{r_p} = \mathbf{A}$.

The adjacency matrix $\mathbf{A}_{AA}$ can be viewed as the two-hop connectivity matrix for A-type nodes, because $\mathbf{A}_{AA} = \mathbf{A}_{AP}\mathbf{A}_{PA}$ in heterogeneous graph is analogous to $\mathbf{A}^2 = \mathbf{AA}$ in homogeneous graph. Equations 5.1 and 5.2 illustrate the relation between **types** in heterogeneous graph and **hops** in homogeneous graph. Therefore, we can view a meta-path as high-order proximity between two nodes.

Theoretical Studies of Neighborhood Aggregation

This section mainly studies the neighborhood aggregation in GNNs-HO and GNNs-HE. A general GNN layer can be defined as:

$$\mathbf{H}^{(k)} = \sigma(\mathbf{A}^p \mathbf{H}^{(k-1)} \mathbf{W}^{(k)}), \tag{5.3}$$

where σ can be any activation function, $\mathbf{H}^{(k-1)} \in \mathbb{R}^{N \times d_{k-1}}$ and $\mathbf{H}^{(k)} \in \mathbb{R}^{N \times d_k}$ is the input and output for layer k, $\mathbf{W}^{(k)} \in \mathbb{R}^{d_k \times d_{k-1}}$ is the trainable transformation matrix.

The neighborhood aggregation iteratively updates the representation of a node by aggregating its neighbors' representations. To mathematically formalize the above insight, the aggregation process can be generically written as follows:

$$\mathbf{s}_i^{(k)} = f_{ag}^{(k)}(\mathbf{h}_j^{(k-1)}, j \in \mathcal{N}_i), \tag{5.4}$$

where $f_{ag}^{(k)}$ is the predefined aggregation function in the k-th layer of a model, $\mathcal{N}_i$ is the defined neighborhood and $\mathbf{s}_i^{(k)}$ is the aggregation result.

One key problem is how to define the neighborhood $\mathcal{N}_i$. In GNNs-HO, $\mathcal{N}_i$ is mainly based on the powers of adjacency matrix $\mathbf{A}^p$. $\mathcal{N}_i$ means one-hop neighbors ($p = 1$) or high-order neighbors ($p > 1$). Once p is fixed, the receptive fields will also be fixed and GNNs-HO aggregates all neighbors within p-hop. In GNNs-HE, neighborhood is defined by the manually defined meta-path as Eq. 5.1. If the meta-path is defined as APA, A-type nodes only aggregate A-type nodes (second-order neighbors), while the P-type (one-hop) neighbors are discarded, resulting in information loss. GTN learns the meta-path by multiplications of softly selected adjacency matrices:

$$\mathbf{A}_R = (\alpha_1 \mathbf{A}_{r_1})(\alpha_2 \mathbf{A}_{r_2})...(\alpha_p \mathbf{A}_{r_p}), \tag{5.5}$$

where $\alpha_1, \alpha_2, ..., \alpha_p$ are learnable parameters who have to be updated in each training epoch, and $\mathbf{A}_R$ in Eq.5.5 also needs to be recalculated in each training epoch. This process is very

time consuming, especially for large scale graphs. Eq.5.5 in GTN defines a general meta-path (receptive field) for all nodes in a graph, which may not be suitable for each node.

5.2.3 A Personalized Receptive Field

For a better neighborhood aggregation, the first desirable is to learn a personalized receptive field (**D1**), then each node can combine information from different hops of neighborhood to assist its representation learning. Based on the theoretical study and analysis, we need a smarter way to combine neighborhood information from different hops. Instead of learning the combination of adjacency matrices Eq. 5.5, we learn a soft combination of the low-dimensional and fix-sized (decided by p) embeddings learned from different hops of neighbors as follows:

$$\mathbf{h}_i^{(k)} = f_{ag}^{(k)}(\alpha_{ir}\mathbf{h}_i^{(k)_r}, r \in (0,1,...,p)) \tag{5.6}$$

where α_{ir} are learnable parameters and $\mathbf{h}_i^{(k)_r}$ is v_i's r-hop representation, $\alpha_{i0} = 1$. Note that $\mathbf{h}_i^{(k)_r}$ contains all information from v_i's r-hop neighbors, not a single neighbor's embedding.

Eq. 5.6 is analogous to the popular receptive module [180] for classic Convolutional Neural Networks (CNN) [107] architectures: it consists of convolutional filters of different sizes determined by the parameter p, where p = 0 (equivalent to transformations of the features in each node without diffusion across nodes in GNN) corresponds to 1×1 convolutions in the receptive module in CNN. The difference is that the convolution filter in CNN convolves all within the $p \times p$ receptive field, and we use α_{ir} to combine different hops of information to define a flexible receptive field for each node (**D1**) within p-hop neighborhood. For example, v_i only aggregates its one-hop and three-hop neighbors, when $p = 3$ and the learned parameters $\alpha_{i0}, \alpha_{i1}, \alpha_{i3} > 0, \alpha_{i2} = 0$.

Two natural follow-up questions are how to learn $\mathbf{h}_i^{(k)_r}$ and α_{ir}. We will give more details in the two following sections.

5.2.4 Hop-aware Projection

Considering different types or hops of neighbors have different traits and their embeddings should fall in different feature space, we design a *hop-specific* transformation matrix ($\mathbf{W}_r$) to project the node features for each hop (or type) of nodes (**D2**). The *r-hop representation* can be expressed as:

$$\mathbf{h}_i^{(k)_r} = \sigma(\mathbf{A}_i^r \mathbf{h}_i^{(k-1)} \mathbf{W}_r^{(k)}), \tag{5.7}$$

where σ can be any activation function, $\mathbf{W}_r^{(k)} \in \mathbb{R}^{d_k \times d_{k-1}}$ is the trainable transformation matrix, $\mathbf{h}_i^{(k-1)} \in \mathbb{R}^{d_{k-1}}$ is the hidden representation of the $(k-1)$-th layer and $\mathbf{A}_i^r$ is the i-th row or line of r-th power of adjacency matrix $\mathbf{A}$. $\mathbf{h}_i^{(k)_r}$ is the *r-hop representation*.

As for heterogeneous graph, *r-hop representation* can also be interpreted as a certain type of neighbors' representation. Because $\mathbf{A}^r$ can be seen as the r-th step in the meta-path, as defined in Def 2.3.1, which also automatically defines the node type (or edge type), as we defined in Def 2.1.3. So, Eq. 5.7 is a general expression and can be used in both homogeneous and heterogeneous graphs to represent a certain hop or type of representation. Note, if there are different types of nodes in the same hop in heterogeneous graph, we use different transformation matrices to learn the embeddings. For example, there are two types of nodes in the one-hop neighbors, and we will use $\mathbf{W}_{11}^{(k)}$ and $\mathbf{W}_{12}^{(k)}$ to transform the two types of neighbors.

5.2.5 Relation-score Learning

After obtaining different hops' embedding, we will illustrate how to calculate $\alpha_{i1}, \alpha_{i2}, \ldots, \alpha_{ip}$ that can define a personalized receptive field as shown in Eq. 5.6. $\alpha_{ir}(r = 1, .., p)$ should be high if the r-hop representation has a close relationship with central node in homogeneous graphs, or r-type neighbors should be considered more. In other words, $\alpha_{i1}, \alpha_{i2}, \ldots, \alpha_{ip}$ reflect the relationship of central node and its different hops neighbors. Therefore, we transfer this problem to how to model the relationship between $\mathbf{h}_i^{(k)_0}$ and $\mathbf{h}_i^{(k)_r}$ (r = 1,2,...,p), which inspires us to apply Knowledge Graph Embedding (KGE) method to solve this problem.

KGE methods aim to model the relationship between *head entity* and *tail entity* in knowledge graph (KG) and assign a score of how likely it is that two entities are in a certain relationship [171, 204]. In our scenario, we define the relation-score as:

Definition 5.2.1. Relation-score. Relation-score is to model the relationship between central node's embedding and a certain hop or type representation, such as $\alpha_{ir} = f_r(\mathbf{h}_i^{(k)_0}, \mathbf{h}_i^{(k)_r})$ is the relation-score of v_i with its r-hop representation.

We want to model the relationship between central node with its p types of representations and usually $p > 1$. We choose four representative KGE models: RESCAL[142], DisMult [217], MLP [44] and NTN [171] with different complexity and expressive powers, which can roughly satisfy different types of datasets. We will introduce them as the complexity from low to high.

In DisMult, each relation is represented as a vector $\mathbf{r}_r$ while it is a matrix $\mathbf{W}_R^r$ in RESCAL[142] to model pairwise interactions between latent factors. For simplicity, we unify the scoring function as:

$$\alpha_{ir} = \mathbf{h}_i^{(k)_0} \mathbf{W}_R^r \mathbf{h}_i^{(k)_r}, \tag{5.8}$$

where $\mathbf{W}_R^r = diag(\mathbf{r}_r)$ in DisMult. [4]

The other two methods MLP [44] and NTN [171] conducts semantic matching using neural network architectures. MLP is a simpler approach and the *head entity*, *tail entity*, *relation* vectors are concatenated in the input layer and mapped to a non-linear hidden layers. The score is then generated by a linear output layer:

$$\alpha_{ir} = \mathbf{W}^{(k)} f([\mathbf{h}_i^{(k)_0} || \mathbf{h}_i^{(k)_r}]), \tag{5.9}$$

where f is the MLP layer and $\mathbf{W}^{(k)}$ is the learnable weight matrix in the linear output layer, $||$ denotes concatenation.

[4] DistMult is actually simplified from RESCAL by restricting $\mathbf{W}_R^r$ to diagonal matrices.

NTN is more expressive that can relate two input entities vectors across multiple dimensions and each slice of the tensor is responsible for one type of entity pair. We compute the relation-scores by the following functions:

$$\alpha_{ir}^{(k)} = \sigma(\mathbf{h}_i^{(k)0}\mathbf{W}_R^{[1:m]}\mathbf{h}_i^{(k)r}), \tag{5.10}$$

where σ is a nonlinear activation function, $\mathbf{W}_R^{[1:m]} \in \mathbb{R}^{d_k \times d_k \times m}$ is a tensor and each slice of the tensor is responsible for instantiation of a relation. $\alpha_{ir}^{(k)} \in \mathbb{R}^m$ is the learned relation-score vector for each relation type, where each entry is computed by on slice of the tensor $\mathbf{W}_R^{[1:m]}$, such as $\alpha_{ir}^{(km)} = \sigma(\mathbf{h}_i^{(k)0}\mathbf{W}_R^{[m]}\mathbf{h}_i^{(k)r})$.

Eq. 5.8, Eq. 5.9 and Eq. 5.10 only need to calculate the relation-scores of $\mathbf{h}_i^{(k)0}$ and $\mathbf{h}_i^{(k)1}, \mathbf{h}_i^{(k)2}, ..., \mathbf{h}_i^{(k)p}$, these low-dimensional embeddings, not central node with all the nodes (N_p is the number of nodes and generally $\mathrm{N}_p \gg p$) within p-hop neighborhood, which is very efficient (**D4**).

5.2.6 Hop-aware Aggregation

After learning the relation-score, we multiply different hops of representations with their corresponding relation-scores, then aggregate them with central node's embedding to get the new embedding $\mathbf{h}_i^{(k+1)}$. This allows for a hop discrimination when central node aggregates different hops' embeddings (**D3**) and can be generally written as:

$$\mathbf{h}_i^{(k)} = f_{ag}^{(k)}(\alpha_{ir}\mathbf{h}_i^{(k)r}, r = 0, 1, .., p), \tag{5.11}$$

where $f_{ag}^{(k)}$ is the predefined aggregation function (aggregator) in the k-th layer of a model, α_{ir} is the relation-score for r-hop embedding of v_i, $\alpha_{i0} = 1$. Analysing α_{ir} can show which hops have an important influence on central node, and this leads to benefits for interpretability (**D5**).

The aggregator in GNNs operates over an unordered set of vectors, because a node's neighbors have no natural ordering. So, the aggregator should be permutation invariant (invariant to permutations of its inputs). Sum, mean, max pooling and concatenation are commonly

used to aggregate the neighbors [72, 215]. While the input, $(\alpha_{i0}\mathbf{h}_i^{(k)_0}, \alpha_{i1}\mathbf{h}_i^{(k)_1}, ..., \alpha_{ip}\mathbf{h}_i^{(k)_p})$, of our aggregator is fixed and ordered (from zero-hop to p-hop), and more complex aggregation functions can be applied here. Besides the four mentioned aggregation functions, we also apply LSTMs [79] in this chapter who are good at dealing with sequential data. Exploring more types of aggregation functions is one of our important future works.

After applying each component introduced in the previous section, we obtain the final node representation, which can be used in different downstream tasks. For multi-class node classification, $\mathbf{H}^{(k)}$ will be passed to a fully-connected layer with a $softmax$ function. The loss function is defined as the cross-entropy error over all labeled examples:

$$\mathcal{L} = -\sum_{l \in \mathcal{V}_l} \sum_{f=1}^{F} \mathbf{Y}_{lf} \ln \mathbf{H}_{lf}^{(K)} , \tag{5.12}$$

where $\mathcal{V}_l$ is the set of node indices that have labels and d_K is the dimension of output equaling to the number of classes. $\mathbf{Y}_{lf} \in \mathbb{R}^{|\mathcal{V}_l| \times F}$ is a label indicator matrix. With the guide of labeled data, we can optimize our model via back propagation and learn the embeddings of nodes and relation-score. The overall process of HHR-GNN in shown in Algorithm 5.1.

5.2.7 Computational Complexity

Two key parts are p-hop representation (Eq. 5.7) learning and relation-score learning (Eq. 5.8, Eq. 5.9 or Eq. 5.10). The powers of the adjacency matrix in Eq. 5.7 can be easily precomputed, because they do not depend on the learnable model parameters, and this effectively reduces the computational complexity of the overall model. Each type or hop of neighbors share the same projection weights and the KGE model is also shared by all nodes in a graph. So, the computation can be parallelized across all nodes. The computational complexity based on different KGE models are summarized in Table 5.2. As for memory requirement, it grows linearly with the size of the dataset and we perform mini-batch training to deal with this issue.

Table 5.2 Comparison of Space Complexity

Methods	Space Complexity
HHR-GNN$_{DisMult}$	$\mathcal{O}(N \times p \times d_k \times d_{k-1} + N \times p \times d_k)$
HHR-GNN$_{RESCAL}$	$\mathcal{O}(N \times p \times d_k \times d_{k-1} + N \times p \times d_k \times d_k)$
HHR-GNN$_{MLP}$	$\mathcal{O}(N \times p \times d_k \times d_{k-1} + N \times p \times 2 \times d_k)$
HHR-GNN$_{NTN}$	$\mathcal{O}(N \times p \times d_k \times d_{k-1} + N \times p \times m \times d_k \times d_k)$

5.2.8 Differences with Existing GNNs

Our model generalizes existing GNNs-HO and GNNs-HE, meanwhile solves some of their remaining issues, such as fix-sized receptive fields, no hop distinction or information loss in the neighborhood aggregation process. We mainly compare our model with GNNs-HO and GNNs-HE from two aspects:

- **The receptive filed.** The general aggregation process and ours are defined in Eq. 5.4 and Eq. 5.11 respectively. The receptive field ($\mathcal{N}_i$) is determined by powers of adjacency matrix (GNNs-HO) or meta-paths (GNNs-He) in Eq. 5.4 and fixed. Some methods also propose more flexible ways to define the receptive filed. GraphSAGE [72] uniformly samples a fixed number of neighbors, FastGCN [36] applies the important sampling with Monte Carlo approaches. Some dropout tricks are proposed. DropEdge [156] randomly removes a set of edges. Instead of random, fixed or important sampling strategies, Graph DropConnect (GDC) [75] jointly learns the edge drop rate at each layer with model parameters, however GDC only models the relationship between the central node with its first-order neighbors without considering the higher-order neighbors. GeniePath [124] adaptively selects a set of significant important one-hop neighbors with adaptive breadth function (attention operator) and filters higher-order neighborhood information with adaptive depth function (LSTM). However, GeniePath does not model the receptive field implicitly, since the predefined p-hop neighbors are all aggregated in the LSTM and it is unclear which hops or types of neighbors are important for the central node's representation learning. Our receptive field in Eq. 5.11 is determined by two factors: p and α_{ir}. p roughly determines the maximum size (how many hops or types) of neighborhood, but which hops or types of neighbors will be

Algorithm 5.1 The overeall process of HHR-GNN

Input: $\mathcal{G}$= $(\mathcal{V}, \mathcal{E}, \mathbf{X})$ with N nodes;

A set of adjacency matrices $\{\mathbf{A}_r\}_{r=1}^{p}$;
Feature matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$;
Labeled nodes $\mathcal{V}_l$;
Label indicator matrix $\mathbf{Y}_{lf} \in \mathbb{R}^{|\mathcal{V}_l| \times F}$;
The number of hops: p;
The number of layers K

Output: The final node embeddings and p-hop relation-scores $\alpha_i^{(K)}$.

Calculate different powers of adjacency matrix.
for k = 1,2,...,K **do**
    for each $v_i \in \mathcal{V}_l$ **do**
        Calculate different hops' representation: $\mathbf{h}_i^{(k)_0}$, $\mathbf{h}_i^{(k)_1}$,..., $\mathbf{h}_i^{(k)_p}$.
        Calculate the relation-scores: $\alpha_{i1}, \alpha_{i2}, ..., \alpha_{ip}$.
        Aggregate different hops representation: $\alpha_{i0}\mathbf{h}_i^{(k)_0}$, $\alpha_{i1}\ \mathbf{h}_i^{(k)_1}$, ..., $\alpha_{ip}\ \mathbf{h}_i^{(k)_p}$.
    end for
end for
Calculate loss: $\mathcal{L} = -\sum_{l \in \mathcal{V}_l} \sum_{f=1}^{F} \mathbf{Y}_{lf} \ln \mathbf{h}_{lf}^{(K)}$.
Back propagation and update parameters in HHR-GNN.

aggregated is determined by the learnable parameter α_{ir}. For example, $p = 3$ means the aggregator allows to aggregate neighborhood information within three-hops. If the final learned relation-scores are $\alpha_{i0}, \alpha_{i1}, \alpha_{i3} > 0, \alpha_{i2} = 0$, the central node actually only aggregates its one-hop and three-hop neighbors. So, the receptive field in our model is automatically learned in an end-to-end fashion for each node, which is more flexible and applicable for both homogeneous and heterogeneous graphs.

- **Aggregation process.** Our aggregation operation differs from others in two aspects, aggregation input and aggregation function. 1) As for the aggregation input, GNNs-HO and GNNs-HE normally aggregate the hidden embeddings of each neighbor $\mathbf{h}_j^{(k-1)}, j \in \mathcal{N}_i$, in the predefined receptive field. These embeddings are unordered and their size is not fixed usually. GeniePath orders these neighbors' hidden representation

as their distance to the central node. While, ours is different hops' representations with their corresponding relation-scores ($\alpha_{ir}\ \mathbf{h}_i^{(k)_r}$, $r \in (0, 1, ..., p)$), which are fixed and ordered (from zero to p-hop). 2) Considering the input data's properties, the aggregation function in GNNs-HO and GNNs-HE should be permutation invariant (invariant to permutations of its inputs), because it operates over a set of unordered vectors. This restricts the application of many powerful tools, such as CNNs [107], LSTMs [79]. On the contrary, there is no restriction of the aggregation function in our model.

5.3 Experiments

5.3.1 Datasets

We conduct semi-supervised node classification experiments on both homogeneous and heterogeneous graphs, which have been widely used in [103, 36, 216, 72, 191, 58, 1, 242]. Homogeneous graphs. We consider two commonly used homogeneous graphs: Cora and Citeseer as in Sec. 3.3.1. For heterogeneous graphs, we use two citation network datasets DBLP and ACM, and a movie dataset IMDB. DBLP contains three types of nodes (papers (P), authors (A), conferences (C)), four types of edges (PA, AP, PC, CP), with research areas of Authors as labels. ACM includes three types of nodes (papers(P), authors (A), subject (S)), four types of edges (PA, AP, PS, SP), with categories of Papers as labels. Each node in DBLP and ACM is represented as bag-of-words of keywords. IMDB consists of three types of nodes (movies (M), actors (A), and directors (D)), and four types of edges (MA, AM, MD, DM) and labels are genres of Movies. Node features are given as bag-of-words representations of plots.

Following [103, 164], we split the Train/Validation/Test for heterogeneous of graphs as shown in Table 5.3. We report the mean accuracy (for homogeneous graphs) and F1-score (for heterogeneous graphs) of 100 runs with random weight initialization for all of our experimental results.

Table 5.3 Overview of the heterogeneous graphs.

Dataset	Nodes	Edges	Features	Classes	Train	Validation	Test	Edge type
DBLP	18,405	67,946	334	3	800	400	2,857	4
ACM	8,994	25,922	1,902	3	600	300	2,125	4
IMDB	12,772	37,288	1,256	3	300	300	2,339	4

5.3.2 Baselines

For homogeneous graphs, we compare our model with four most related methods, GCN [103], GAT [191], GraphSAGE-mean [72] and MixHop [1]. we keep the same architecture as the original papers. We use the results from Chapter 3 and Chapter 4 for NFC-GCN and LA-GCN.

For heterogeneous graphs, we compare with conventional network embedding methods: DeepWalk [151], metapaht2vec [46], and GNN-based methods: GCN [103], GAT [191], HAN [206] and GTN [164], following [164].

- **Conventional methods.** We apply DeepWalk directly to the heterogeneous graphs and ignore the node (edge) types. metapath2vec applies the meta-path guided random walks to model the context of a a given node.

- **GNN-based method.** GCN and GAT are originally designed for homogeneous graphs. To apply them on heterogeneous graphs, we ignore node (edge) types and directly perform these methods. HAN aggregates the neighbors from the meta-path and applies both node-level and semantic-level attentions. GTN learns a soft selection of edge types for generating multiple meta-paths and applies the neighborhood aggregation stratety to learn a given node's representation.

5.3.3 Hyperparameters Setting

We model the central node's representation with its one-hop and two-hop neighbors for Cora and Citeseer. Throughout experiments, we use the Adam optimizer [102] with learning rate in the set $\{0.002, 0.004, 0.006, 0.008\}$, regularization parameter $\in \{5\times10^{-3}, 5\times10^{-4}, 5\times10^{-5}\}$, the dropout rate $\{0.2, 0.4, 0.6\}$. We train all models for a maximum of 500 epochs

and apply early stopping with a patience of 20. We use two HHR-GNN layers for Cora, in which the dimension of two GNN projection layers is 32 and 8 respectively, and two NTN ($32\times$ 32 $\times$ 4, 8 $\times$8 $\times$4) are incorporated to learn the relationships between central node with its one-hop and two-hop neighbors respectively. For Citeseer, we use one HHR-GNN layer and set the projection dimension as 32. We try different aggregators (concatenation, sum, average, max pooling, LSTM and GRU with hidden dimension of 16). As for the GCN, GAT, GraphSAGE and MixHop baselines, we keep the same architecture as the original papers. [5]

The selected three heterogeneous graphs include three types of nodes and neighbors within two-hop covers all three types. Thus, we model the relationships between a central node with its one-hop and two-hop neighbors: AP, AC (A-P-C), AA (A-P-A) for DBLP; PA, PS, PP (P-A-P, P-S-P) for ACM; MA, MD, MM (M-A-M, M-D-M) for IMDB, based on the tasks. It should be emphasized that in heterogeneous graphs, we use different weight matrices to learn embeddings for different types of nodes in the same hop. For example, in ACM, P has two types of neighbors (A and S) in one-hop neighborhood and we use two weight matrices to learn their respective node embeddings. We employ the same aggregators as those for homogeneous graph and the hidden layer dimension is 32. The other parameters, such as dropout rate, weight decay and learning rate, follow the same setting as for homogeneous graphs.

5.3.4 Performance for Node Classification

Results of node classification on homogeneous and heterogeneous graphs are summarized in Table 5.4 and Table 5.5 respectively. We observe that our model achieves the best performance on all the datasets. This indicates that our method is very competitive on both homogeneous and heterogeneous graphs.

For homogeneous graphs, our method leverages an enlarged (two-hop) neighborhood and meanwhile learns an personalized receptive field for each node, which can provide more useful information for learning central node's representation. This is especially beneficial

[5] With no specific notification, we will use NTN and concatenation as our KGE models and aggregation function.

Table 5.4 Node classification for homogeneous graph (%) (mean ± 95% confidence interval over 100 runs)

. The best results are in **bold** and the second best ones are underlined.

	Methods	Cora	Citeseer
Baselines	GCN	88.0 ± 0.235	77.8 ± 0.316
	GAT	80.4 ± 0.255	75.7 ± 0.431
	GraphSAGE	82.2 ± 0.135	71.4 ± 0.605
	MixHop	88.3 ± 0.446	-
HHR-GNN$_{NTN}$	Con	**88.84 ± 0.280**	77.28 ± 0.318
	Sum	87.59 ± 0.204	76.60 ± 0.416
	Aveg	88.45 ± 0.204	75.72 ± 0.311
	Max pooling	87.84 ± 0.335	**78.54 ± 0.302**
	LSTM	85.39 ± 0.280	74.74 ± 0.233
	GRU	85.63 ± 0.335	75.34 ± 0.318

Table 5.5 Node classification for heterogeneous graph (F1-score) (mean ± 95% confidence interval over 100 runs)

. The best results are in **bold** and the second best ones are underlined.

	Methods	DBLP	ACM	IMDB
Baselines	DeepWalk	63.2 ± 0.235	67.4 ± 0.329	32.1 ± 0.229
	M2vec	85.5 ± 0.195	87.6 ± 0.555	35.2 ± 0.367
	GCN	87.3 ± 0.335	91.6 ± 0.305	56.9 ± 0.329
	GAT	93.7 ± 0.370	92.3 ± 0.140	58.1 ± 0.487
	HAN	92.8 ± 0.127	90.9 ± 0.191	52.3 ± 0.194
	GTN	94.2 ± 0.762	92.7 ± 0.381	60.9 ± 0.623
HHR-GNN$_{NTN}$	Con	94.7 ± 0.167	**94.8 ± 0.217**	60.7 ± 0.334
	Sum	92.1 ± 0.163	93.3 ± 0.203	**61.09 ± 0.325**
	Aveg	94.3 ± 0.211	93.6 ± 0.270	59.8 ± 0.297
	Max pooling	94.8 ± 0.403	92.1 ± 0.386	58.7 ± 0.529
	LSTM	95.1 ± 0.277	93.2 ± 0.335	59.2 ± 0.359
	GRU	**95.3 ± 0.309**	93.4 ± 0.410	59.1 ± 0.379

for sparse graphs. [6] Graph-SAGE and Mix-Hop can leverage higher-order neighborhood information, but they can not treat different hops of neighbors differently. NFC-GCN and LA-GCN get better results, because they can utilize the feature-level local information

[6]The average node degree for Cora and Citeseer are 4.9 and 3.7 respectively.

sufficiently, and this mechanism is suitable for homogeneous graph representation learning, especially for graphs with node attributes.

For heterogeneous graphs, the results of other methods come from [164]. [7] The results of DeepWalk, GCN and GAT reveal that treating heterogeneous graphs as homogeneous graphs is not an optimal choice. It can be seen in Table 5.5, our method achieves the best performance on all datasets. Our model outperforms GTN by about 1.1%, 0.98 %, 1.5% and HAN by around 2.3%, 2.7%, 9.3% on DBLP, ACM and IMDB respectively. We think the improvement is caused by that our model provides a personalized context for each node and utilizes different types of neighbors to learn the new representation for a central node. HAN utilizes manually designed meta-paths to transform a heterogeneous graph to a homogeneous graph, so that the information from some types of nodes would be lost. This could damage the central node's representation learning and make the performance unstable. The key idea of GTN is to learn a general meta-path, while this path is not suitable for every node in the graph. Our method considers all types of nodes within a fixed-hop neighborhood, meanwhile treats different types of nodes differently in aggregation. Compared with HAN and GTN, our method can provide a customized receptive field for each node and absorb useful information from all types of nodes.

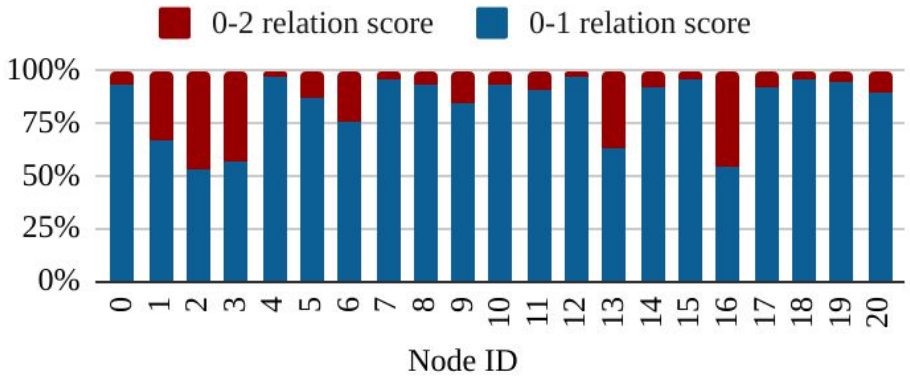
5.3.5 Interpretability Study

A key part of our model is the learned relation-score that can be used to determine each node's receptive field (**D1**) and guide the hop-aware aggregation (**D2**), meanwhile provide a qualitative and quantitative understanding of the relationship between a central node and its different hops or types of neighbors. In this section, we aim to answer the following **Q**uestions: (1) what we expected, (2) what we learned and (3) what we concluded.

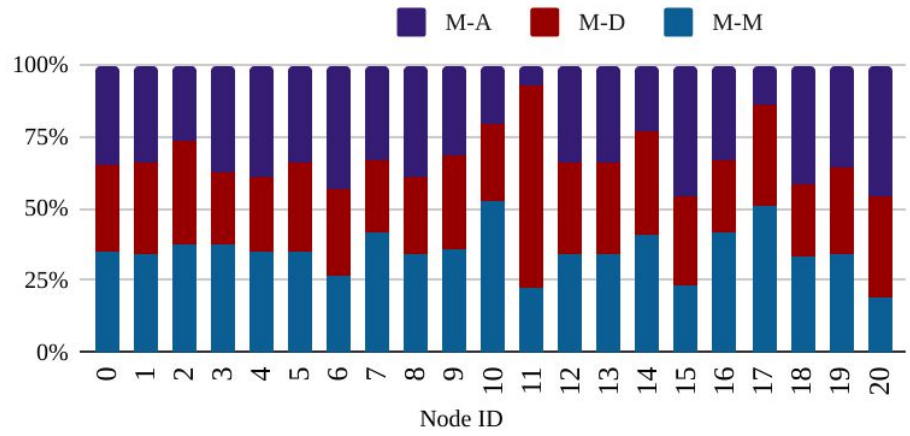
For **Q1**, the expectation intuitively is that the important hops or types' representation should have higher relation scores.

In order to answer **Q2**, we first visualize the leaned one-hop (0-1) and two-hop (0-2) relation-scores of the first 21 nodes on Cora, as shown in Fig. 5.2a. On IMDB, we show

[7] Following [164], we do not show the standard deviation of our results.



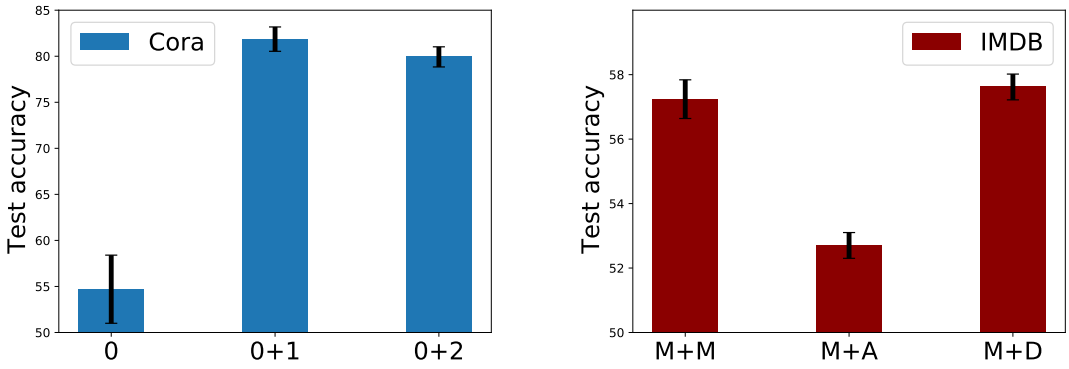
(a) Cora.



(b) IMDB

Fig. 5.2 (a) $0-1$ relation-scores and $0-2$ relation-scores of 21 nodes on Cora. (b) MA, MD and MM relation-score of 21 nodes on IMDB.

the learned relation-scores of the first 21 nodes for different types of neighbors, including MA, MD (two types of one-hop neighbors) and MM (two-hop neighbors formed by M-A-M and M-D-M) in Fig. 5.2b. For a better comparison, we use the 100% stacked column chart and a larger proportion means a higher relation-score in each bar. Fig. 5.2a shows that 0-1 relation-scores are generally higher than 0-2 relation-scores, which means directly linked (one-hop) neighbors have a closer relationship than indirectly connected (two-hop) neighbors. Compared with Fig. 5.2a, the relation-scores in Fig. 5.2b look much more complex, due to the intrinsic fact of heterogeneous graphs that each central node is connected with different types of neighbors. From Fig. 5.2b, we can observe that each node has their respective



(a) Cora (b) IMDB

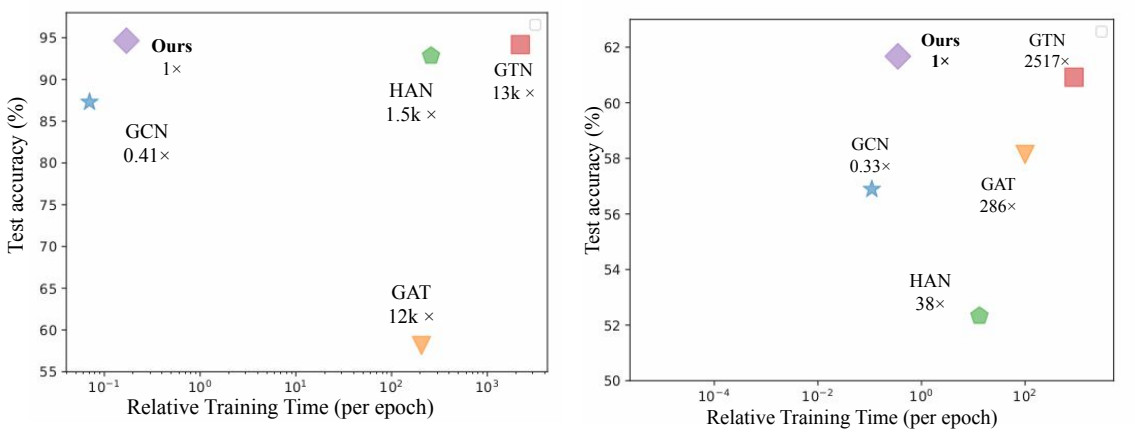
Fig. 5.3 Node classification results on Cora (a) and IMDB (b) by aggregating a central node with a specific hop or type of neighbors.

preference for the different types of nodes, e.g., Node 11 has a much closer relationship with D (the red part takes a big proportion in the bar), while Node 10 with M.

The above observations indicate that our model learns distinct relation-scores as expected and can help the aggregation process focus on important hops or types of neighbors (**Q3**). To further verify the relation-scores are reasonable in Fig. 5.2a and Fig. 5.2b, we conduct node classification tasks on Cora and IMDB without learning relation-scores. Concretely, we equally aggregate a central node's representation with a specific hop or type of neighbors. [8] The results are summarized in Fig. 5.2. By analyzing the results from Fig. 5.3a, we can conclude that one-hop neighbors are generally more important than two-hop neighbors on central node's representation learning, which explains well why our learned 0-1 relation-scores are generally higher than 0-2 scores in Fig. 5.2a. Fig. 5.3b shows that M+D [9] obtains the similar accuracy with M+M, which confirms that different types of neighbors in heterogeneous graphs also benefit central nodes' representation learning and should not be discarded as done in previous GNNs-HE. Instead, our method considers all types of neighbors based on leaned adaptive relation-scores as shown in Fig. 5.2b, which is more reasonable.

[8] 0+1 means a central node $\mathbf{h}_i^{(k)_0}$ only aggregates its one-hop representation $\mathbf{h}_i^{(k)_1}$.

[9] M+D mean M-type nodes only aggregate D-type of neighbors, M+M, M+A are with the similar definitions.



(a) DBLP (b) IMDB

Fig. 5.4 Performance over training time on DBLP and IMDB.

5.3.6 Efficiency Study

In Fig. 5.4a and Fig. 5.4b, we plot the performance of state-of-the-art GNNs-HE and our method over their training time on two large graphs: IMDB and DBLP. Our model achieves competitive performance in both accuracy and efficiency.

As shown in Fig. 5.4a and Fig. 5.4b, GCN is the most efficient method, but can not ensure the accuracy. This is because simply aggregating the neighbors is not suitable for heterogeneous graphs that contain more complex neighborhood information than homogeneous graphs. Compared with other methods, GTN has an obvious advantage in accuracy, while does not perform well in efficiency. We think the reason is that GTN aims to learn an optimal meta-path by utilizing a 1×1 convolution to softly select adjacency matrices, which makes the final adjacency matrix very dense and the later computations (multiplication of the selected matrices and GCN operation) very time-consuming, especially for large graphs. While our method learns the meta-path by utilizing the low-dimension hidden representations and a light-weight NTN model, which is very efficient, especially for heterogeneous graphs with a few types of nodes. For heterogeneous graphs with more types of nodes and relations, our model needs more mapping matrices and slices of the NTN model, and this will slow the computation, which we will explore in the future works.

Table 5.6 Node classification results of different KGE models. No_rs means no relation-score (%) (mean ± 95% confidence interval over 100 runs).

	No_rs	DisMult	RESCAL	MLP	NTN
Cora	81.59±0.22	**84.17±0.53**	82.66±0.47	83.32±0.34	83.84±0.29
Citeseer	70.24±0.39	**72.51±0.42**	73.16±0.34	70.56±0.41	72.28±0.32
IMDB	57.65±0.43	**61.14±0.33**	60.93±0.24	60.8±0.32	60.67±0.41
ACM	90.75±0.32	93.52±0.26	94.60±0.27	93.78±0.24	**94.83±0.30**
DBLP	89.81±0.22	94.22±0.19	**95.03±0.33**	93.89±0.41	94.65±0.42

5.3.7 Different KGE Model Study

Different KGE models can be applied to learn the relation-scores between a central node and its different hops' embeddings. The complexity of four mentioned models: DisMult, RESCAL, MLP, NTN, are show in Table 5.2. DisMult is the simplest KGE method and NTN is the most expressive model with the most parameters to learn. We compare the performance of these methods in downstream node classification tasks. We fix the aggregation function as concatenation and try different KGE models to learn the relation-scores for node classification tasks. Results are summarized in Table 5.6

Table 5.6 reveals that more expressive models (RESCAL and NTN) do not lead to better performance. A possible reason is that the training samples are quite limited, especially for Cora and Citeseer, there are only 140 and 120 training samples respectively. In these scenarios, DisMult with less parameters to learn could be a good choice. With more training samples (DBLP and ACM with 800 and 600 training samples), NTN or RESCAL could be considered first.

5.4 Summary

In this chapter, we proposed Hop-Hop Relation-aware Graph Neural Networks (HHR-GNN) [234], a new class of Graph Neural Networks, to solve the second research question: hop-level attention of local information. HHR-GNN leveraged knowledge graph embedding techniques

to learn the relation-score between central node and its different hops or types of neighbor. The learned relation-score can be used to define a personalized receptive field for each node and hop-aware aggregation to distinguish different hops or types of neighbors. What's more, our proposed framework is very flexible and applicable and different KGE methods and aggregation functions can be applied based on different datasets. We evaluated HHR-GNN against state-of-the-art GNNs on node classification task. Experimental results showed that HHR-GNN is competitive no matter in accuracy and efficiency. Besides, it can identify the useful and personalized context for each node, which leads to benefits in interpretability and provides insights on the effective hops or types of neighbors for prediction. Different from NFC-GCN and LA-GCN in Chapter 3 and Chapter 4, HHR-GNN can explore higher-order neighborhood information and is suitable for both homogeneous and heterogeneous graphs, meanwhile solves some of their remaining issues of current GNNs-HO and GNNs-HE, such as fix-sized receptive fields, no hop distinction or information loss in the neighborhood aggregation process.

Part II

Applications

Chapter 6

Multi-Task GNN for Personalized Video Search

6.1 Introduction

With the popularity of smart phones and advancement of communication technologies, people can easily record and edit videos. Everyday, billions of new videos are created, shared and watched [227]. Video search provides a convenient entry point for customers to browse and watch videos from numerous videos. Different from Web search, video search is mainly for entertainment and users' preferences could be very diverse because of their different backgrounds. For example, when issuing the query "*Welcome to New York*", a music fan may want to find the music video from *Taylor Swift*, while a film fan may want to watch the film directed by *Abel Ferrara*. Similar to many other vertical search areas, there is a strong need to return personalized search results to different users for the same query in video search.

The user click information is often used to train personalized search models (PSMs), while the click signal in video search may not necessarily indicate relevancy between the query and video [247]. Statistics show that clicks of irrelevant videos can be as high as 30% of all clicks in our system. This may be due to the characteristics of video search, including: users tend to have more time when they are browsing videos; videos serve more entertainment purpose than non-video documents; videos contain richer information than non-

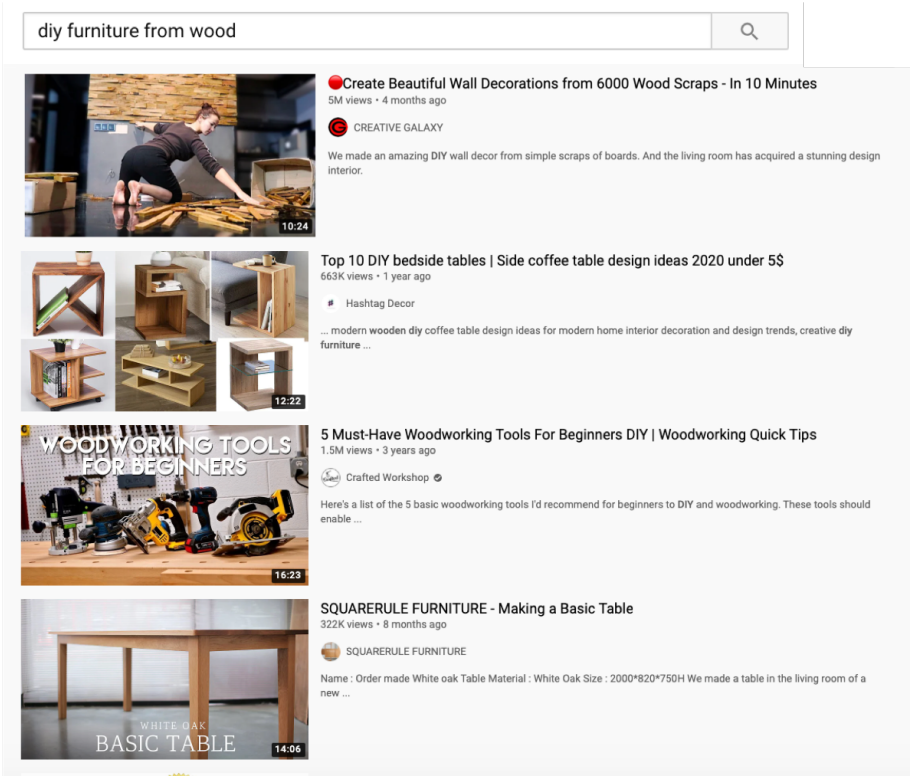


Fig. 6.1 Example top ranked results for the query “diy furniture from wood": the first and third videos are not relevant to the query, but may still be interesting to the user.

video documents. Therefore, users can be easily attracted to interesting but irrelevant videos. Figure 6.1 shows a list of top videos returned from YouTube given the query “diy furniture from wood". We can see that the first video (showing the wall decoration) and the third video (showing the woodworking tools) are not very relevant to the query. Given this list of videos, however, a user who is currently doing some home improvement projects, may also be interested in furnishing the walls and clicking on the first video, or be interested in checking out the tools and clicking on the third video. Therefore, we propose that both customized search results and query-document relevance should be considered in personalized video search, which have rarely been exploited by current PSMs.

The most common paradigm of existing PSMs is to apply deep learning to learn the semantic similarity of query-document pair and personalize the search results by considering users’ information, i.e., users’ location, meta information, social connections [221, 87, 88] or search history [4, 60, 126, 249]. While, the data in the video search is usually too sparse to

train a good PSM [83, 197]. Especially when queries are short and vague, it is more difficult for PSMs to learn accurate representations. Additional information such as the relationship among users, queries in user-query, click graphs respectively can provide rich information beyond textual and video content [38, 128, 23], which have rarely been exploited by current PSMs.

While intuitively useful to integrate similar users and click graph information into personalized video search, and graph neural networks (GNNs) [103, 73] can be applied to model the topological information of graph datasets. However, two unique challenges arise in achieving this goal in our scenario. (i) The graphs are heterogeneous and contain millions of nodes and edges. How to design an efficient GNN architecture for real-world graphs is the first obstacle we need to overcome. (ii) The user-query graph and query-document click graph in the real industry system are stored as user-query and query-document pairs, rather than entire graphs used in typical GNNs, which could prevent the message passing between different hops of neighbors. For a given user, both the local information (issued queries) and higher-order neighbors, such as similar users from the second-hop neighbors (user-query-user) in the user-query graph are important for the user's representation learning. Hence, how to jointly capture the local as well as the higher-order neighborhood information remains a significant challenge.

In light of the aforementioned motivations and challenges, we propose a GNN-based multi-task learning framework for personalized video search [236] where two bipartite graphs: user-query graph and query-document [1] click graph are integrated into the learning process, in addition to the semantic representations learned from text (query and video title) and video content with BERT [43] and Two-Stream Inflated 3D ConvNet (I3D) [30]. To efficiently utilize the graph information, we perform the graph convolution by sampling fixed-size neighbors from graphs and alleviate the need to operate GNN on the entire graph during training. To utilize different hops of neighbors, we propose a hierarchical GNN architecture to simultaneously capture both local and higher-order interactions among nodes. It learns user representations from their issued queries (users' first-hop neighbors), neighboring users

[1] In our section, document equals to video.

(user's second-hop neighbors) in the user-query graph. Query representations are learned from clicked videos and neighboring queries in the query-document click graph. Document representations are learned from their associated queries and neighboring videos in the query-document click graph. Considering the heterogeneity of the used information, we design a hop-specific transformation strategy, which enables nodes to treat different hops of neighbors differently. Assuming the click task and query-video relevance task are closely related, therefore we propose to model the two tasks in a commonly used multi-task learning framework.

In summary, our main contributions are summarized as follows:

- We design an efficient GNN-based multi-task learning framework for real industry personalized video search. We utilize two bipartite graphs: user-query and query-document graphs to enrich the representation for users, queries and videos. To the best of our knowledge, this is the first attempt to apply graph information and GNN for personalized video search.
- In real industry system, we identified that the click signal may indicate attractiveness but not necessarily indicate relevance. Different from other PSMs trained only by click label, our model also considers the relevance between queries and videos.
- We conduct extensive experiments on a large-scale real dataset obtained from a well-known video search platform. Experimental results show that our proposed model can significantly outperform most state-of-the-art PSMs.

6.2 Methodology

In this section, we introduce our GNN-based multi-task learning framework for personalized video search, which mainly consists of two key part: 1) semantic representation learning that focuses on text representation learning (query and video title) and video representation learning; 2) graph representation learning that utilizes the hierarchical GNN architecture to

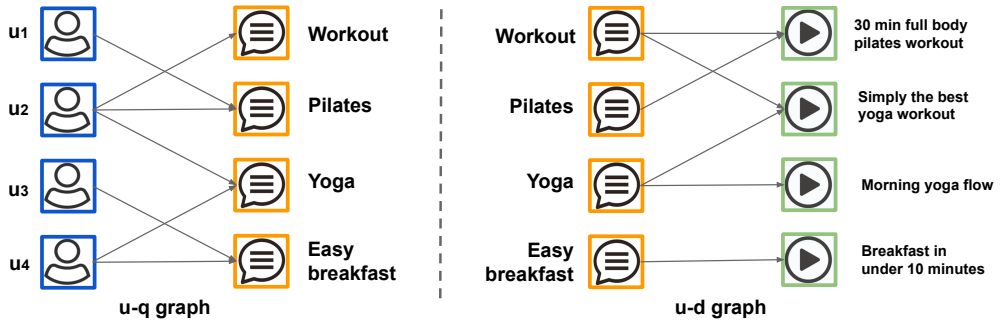


Fig. 6.2 User-query graph and query-document click graph. In user-query graph, nodes are users and queries, and edges mean users issued queries. Query-document graph contains two types of nodes: queries and documents and links mean clicks for query-document pairs by any user.

leverage user-query graph and query-document graph to learn better user, query and video representations.

6.2.1 Problem Definition

For the personalized video search task, we first formulate our problem as follows. When a user u issues a query q, the video search engine is required to retrieve the most relevant videos as a ranking list. Through re-ranking the unpersonalized list for different users according to their interests, backgrounds, the video search engine finally provides a personalized ranking list to each individual user.

Considering the sparsity of users' search history, vague and short queries, we leverage extra information from the click-through data to learn their better representations. The click-through data contains both the user search behaviours and user click-through behaviors, thus we utilize these information to construct two bipartite graphs: user-query graph $\mathcal{G}_{uq}$ and query-document graph $\mathcal{G}_{qd}$, as shown in Fig. 6.2. In addition to the text and video information, our goal is to leverage graph information $\mathcal{G}_{uq}$ and $\mathcal{G}_{qd}$ to learn high-quality embeddings. We assume that the click task and the query-video relevance task are closely related. Thus for a given triple (*user, query, video*), we share the query and video's representations for the two tasks and apply two independent neural networks for the click score and query-video relevance score estimation respectively.

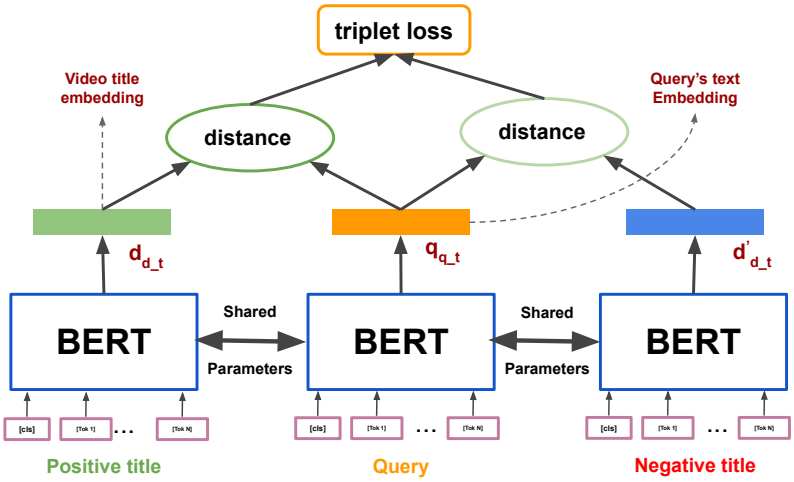


Fig. 6.3 Query-Title Matching Model: train BERT with the triplet loss to get query text embedding $\mathbf{q}_{q\_t}$ and video title embedding $\mathbf{d}_{d\_t}$.

6.2.2 Semantic Representation Learning

For a given triple <u_u, q_q, d_d>, we firstly utilize text (query and video title) and video information to learn the semantic representations of q_q and d_d.

Inspired by the great success achieved by the large-scale pre-trained transformer-based language models, such as BERT [43], we design a BERT_based Query-Title Matching Model (QTMM) to obtain the embeddings of queries and video titles, as shown in Fig. 6.3. The input of the model includes three components: a *query*, a *positive title*, and a *negative title*, where *positive title* is the title of a clicked video given the query q_q and *negative title* is chosen randomly from the unclicked videos for q_q. A special token $[CLS]$ is attached at the beginning of each input component, which can aggregate the sequence information to generate embeddings during learning. Then, three multi-layer Transformer BERT with shared parameters are adopted to capture the contextual information in the text and generate the embeddings of *query*, *positive title* and *negative title* ($\mathbf{q}_{q\_t}$, $\mathbf{d}_{d\_t}$, $\mathbf{d}'_{d\_t}$). We train QTMM with the triplet loss [162] as following:

$$\mathcal{L}_t = max(d_{qp} - d_{qn} + margin, 0), \tag{6.1}$$

where d_{qp} is the euclidean distance of $\mathbf{q}_{q\_t}$, $\mathbf{d}_{d\_t}$ and d_{qn} is the distance of $\mathbf{q}_{q\_t}$, $\mathbf{d}'_{d\_t}$.

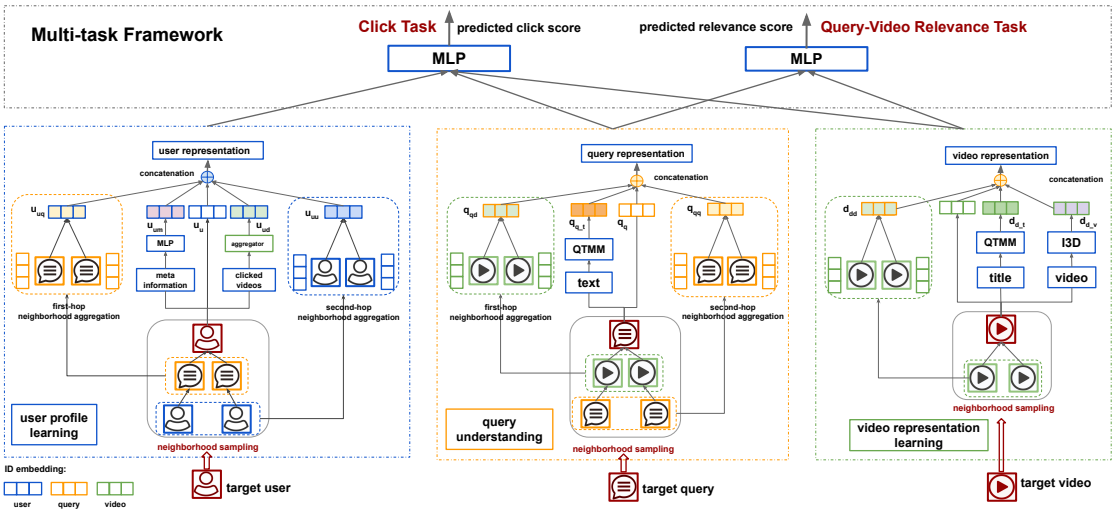


Fig. 6.4 An illustration of our GNN-based multi-task framework. Given the triple <u_u, q_q, d_d>, we first apply the QTMM and I3D to learn the semantic representations of q_q and d_d. Then, we sample fixed-size neighbors for u_u, q_q, d_d from the u-q and q-d graphs and leverage the graph information with the proposed hierarchical GNN architecture simultaneously capturing both local and higher-order interactions among nodes to enhance their representation. Finally, we combine representations learned from text, video and graph for the click task and query-video relevance task for personalized video search.

To learn the video representation $\mathbf{d}_{d\_v}$, we apply the similar strategy as in the text representation learning. Given the triple <*video, positive query, negative query*>, we feed the query to BERT and video to Two-Stream Inflated 3D ConvNet (I3D) [30], a popular model for video representation learning and train them with the triplet loss. To learn better semantic representations of queries and videos, both QTMM and I3D are trained on billions of the construct triples.

6.2.3 Graph Representation Learning

Beyond text and video information, we aim to apply GNNs to leverage graph information to alleviate the sparsity issue, as shown in Fig. 7.3. There are mainly two steps: 1) neighborhood sampling which samples fixed-size neighbors (from different hops in a graph) for a given node; 2) hierarchical aggregation that utilizes different hops of neighborhood information to enrich a given node's representation learning.

Neighborhood Sampling. GNN consists of two key steps: neighborhood aggregation and feature transformation. So, we start from how we define the neighborhood $\mathcal{N}_i$ in the real industrial graphs, an important innovation of our approach.

For example, users' information demand can be clearly revealed by their issued queries (first-hop neighbors from $\mathcal{G}_{uq}$). Besides, the relationship among users (second-hop neighbors from $\mathcal{G}_{uq}$ who issued the similar set of queries) should not be ignored, which can be extremely helpful to overcome the sparsity problem of users' history. Thus both the first and second-hops of neighbors should be considered in the neighborhood aggregations process.

Conventional GNNs can leverage k-hop neighborhood information by stacking k GCN layers or perform random walks on the graph [103, 73, 216, 105]. However, the user-query graph and query-document click graph in the real industry system are stored as user-query and query-document pairs, rather than the entire graphs used in typical GNNs, which prevents the message passing between different hops of neighbors and the utilization of random-walk strategy. Besides, some hot queries and documents have tons of neighbors. Considering the memory and efficiency problem, we sample a fixed-size neighbors for each user, query and video from their first-hop (user-query pairs) and second-hop ((user-user pairs) neighborhood. For u_u, the sampled neighbors are issued queries (first-hop neighbors) $\mathcal{N}_{u_u\_q} = [q_{u1}, q_{u2}, ..., q_{uK}]$ and similar users $\mathcal{N}_{u_u\_u} = [u_{u1}, u_{u2}, ..., u_{uK}]$. Considering similar queries and clicked videos for a given query in the query-document graph can be exploited to enrich the current query and provide more search context to help disambiguation, especially for short and ambiguous queries, we sample K clicked videos $\mathcal{N}_{q_q-d} = [d_{q1}, d_{q2}, ..., d_{qK}]$ and similar queries $\mathcal{N}_{q_q-d} = [q_{q1}, q_{q2}, ..., q_{qK}]$ from $\mathcal{G}_{qd}$. Videos sharing many co-clicked queries should also be close in the vector space and we sample a fixed-size videos $\mathcal{N}_{d_d-d} = [d_{d1}, d_{d2}, ..., d_{dK}]$ from d_d's second-hop neighbors (clicked videos with same queries).

Hierarchical Aggregation. After getting sampled neighbors for users, queries and videos, a natural idea is to aggregate their neighborhood information to enrich their representations.

The conventional neighborhood aggregation in GNNs is

$$\mathbf{h}_i^{'} = f_a(\mathbf{h}_j, j \in \mathcal{N}_i), \tag{6.2}$$

where f_a is a predefined aggregation function (aggregator) and $\mathcal{N}_i$ is the neighborhood. In feature transformation stage, the central node $\mathbf{h}_i$ first combines with $\mathbf{h}_i^{'}$, followed by a linear mapping or MLPs to get its new representation.

Different from conventional GNNs, the sampled neighbors ($\mathcal{N}_i$) contain both homogeneous and heterogeneous neighbors for users and queries. Such as $\mathcal{N}_{u_u\_q}$ and $\mathcal{N}_{u_u\_u}$ are heterogeneous and homogeneous neighbors for u_u respectively. A naive approach is to ignore the node/edge types and treat them as in a homogeneous graph. [2] This, apparently, is suboptimal since different types of neighbors have different traits and their embeddings should fall in different feature spaces. Thus we design a hierarchical aggregation strategy and apply different aggregation functions to aggregate a given node's first-hop and second-hop neighbors respectively.

Take user u_u in the user-query graph as an example, we apply two different aggregation functions: f_{uq} f_{uu} to learn user's representations by leveraging its first-hop (queries) and second-hop (similar users) neighborhood information respectively, as shown:

$$\mathbf{u}_{uq} = f_{uq}(\mathbf{q}_{ui}, ui \in \mathcal{N}_{u_u\_q}). \tag{6.3}$$

$$\mathbf{u}_{uu} = f_{uu}(\mathbf{u}_{ui}, ui \in \mathcal{N}_{u_u\_u}), \tag{6.4}$$

where f_{uq} and f_{uu} are query-type and user-type aggregators that focus on aggregating homogeneous and heterogeneous information respectively. A natural follow-up question is how to design the aggregation function.

Ideally, the aggregation function would be symmetric (i.e., invariant to permutations of its inputs), which ensures our model can be applied to arbitrarily ordered neighbors [215, 73]. [3] Take f_{uq} as an example, candidate aggregation functions can be sum (f_{uq_s}), mean (f_{uq_m}), maxpooling, ($f_{uq_{mp}}$) [4] and attention ($f_{uq_{att}}$) aggregation [190, 191] as shown:

[2] A graph with one type of nodes and edges

[3] There is no order between $\mathbf{u}_{u1}$, $\mathbf{u}_{u2}$... $\mathbf{u}_{uK}$. A user issued queries, $\mathbf{q}_{u1}, \mathbf{q}_{u2}, ..., \mathbf{q}_{uK}$ can be sequential datasets if we have their timestamp information.

[4] f_{uq_s} means summation of the embeddings of neighbors, f_{uq_m} means average of these embeddings. $f_{uq_{mp}}$ applies max-pooling operator to each of the computed feature.

$$\mathbf{u}_{uq} = f_{uq_{att}}(\mathbf{q}_{ui}) = \sum_{i=1}^{K} \alpha_{ni}\mathbf{q}_{ui}(ui \in \mathcal{N}_{u_u\_q}), \tag{6.5}$$

where α_{ni} can be learned from

$$\alpha_{ni} = \frac{exp(LeakyReLU(\mathbf{W}_u[\mathbf{u}_u \parallel \mathbf{q}_{ui}] + \mathbf{b}_u))}{\sum_{d=1}^{K} exp(LeakyReLU(\mathbf{W}_u[\mathbf{u}_u \parallel \mathbf{q}_{ud}] + \mathbf{b}_u)}, \tag{6.6}$$

where $\mathbf{W}_u$ and $\mathbf{b}_u$ are trainable parameters in the attention network, $\parallel$ denotes concatenation.

Similarly, for a given query q_q, we apply the hierarchical aggregation strategy and utilize different aggregation functions f_{qd} and f_{qq} to aggregate its first-hop neighbors $\mathcal{N}_{q_q\_d}$ (clicked videos) and second-hop neighbors $\mathcal{N}_{q_q\_q}$ (similar queries) in the query-document graph to obtain $\mathbf{q}_{qq}$ and $\mathbf{q}_{qd}$ respectively. This can provide more search context to help disambiguation, especially for short and ambiguous queries.

Videos sharing many co-clicked queries should also be close in the vector space. For the given video d_d, We apply f_{dd} as the aggregator to aggregate its second-hop neighbors $\mathcal{N}_{d_d\_d}$ as following:

$$\mathbf{d}_{dd} = f_{dd}(\mathbf{d}_{di}, di \in \mathcal{N}_{d_d\_d}). \tag{6.7}$$

6.2.4 Incorporating User Meta Information

Besides the user-query graph information, we also have users' additional features and search history information. To better characterize users and retrieve personalized search results, we augment our model with additional user features which are represented in a multi-field multi-hot encoding form. Each field contains multiple discrete categorical features, such as gender, job, position, which are translated into several high-dimensional sparse features via one-hot encoding. For example, [*gender=female, job=teacher*] can be represented as:

$$\underbrace{[1,0,0]}_{gender}\underbrace{[0,1,...,0]}_{job}. \tag{6.8}$$

Then the raw sparse feature $\mathbf{u}_{u_{ms}}$ is fed into the MLPs to generate low-dimensional real-valued dense vector $\mathbf{u}_{um}$:

$$\mathbf{u}_{um} = MLP(\mathbf{u}_{u_{ms}}). \tag{6.9}$$

User clicked videos can directly reflect users' preference, such as preferred video type, watching habits (prefer long or short video) and so on. Applying the similar way as learning $\mathbf{u}_{uq}$ and $\mathbf{u}_{uu}$, we get $\mathbf{u}_{ud}$ as following:

$$\mathbf{u}_{ud} = f_{ud}(\mathbf{d}_{ui}, i = 1, 2, ..., K), \tag{6.10}$$

where $\mathbf{d}_{ui}$ is the ID embedding from video ID embedding matrix $\mathbf{D} \in \mathbb{R}^{N_q \times D}$.

6.2.5 Ranking Score Generation

Finally, we concatenate u_u's ID embedding $\mathbf{u}_u$, embeddings learned from meta-information $\mathbf{u}_{um}$, u-q graph $\mathbf{u}_{uq}$, $\mathbf{u}_{uu}$ and clicked videos $\mathbf{u}_{ud}$ to get its new embedding $\mathbf{u}_u^{'}$

$$\mathbf{u}_u^{'} = \mathbf{u}_u \oplus \mathbf{u}_{um} \oplus \mathbf{u}_{uq} \oplus \mathbf{u}_{uu} \oplus \mathbf{u}_{ud}, \tag{6.11}$$

where $\oplus$ is the operation of vector concatenation.

For query q_q, we concatenate its ID embedding $\mathbf{q}_q$, text embedding $\mathbf{q}_{q\_t}$ and embeddings $\mathbf{q}_{qd}$ and $\mathbf{q}_{qq}$ learned from q-d graph to get its new embedding $\mathbf{q}_q^{'}$

$$\mathbf{q}_q^{'} = \mathbf{q}_q \oplus \mathbf{q}_{q\_t} \oplus \mathbf{q}_{qd} \oplus \mathbf{q}_{qq}. \tag{6.12}$$

For video d_d, we concatenate its ID embedding $\mathbf{d}_d$ and, semantic embeddings learned from text and video $\mathbf{d}_{d\_t}$, $\mathbf{d}_{d\_v}$ and $\mathbf{d}_{dd}$ learned from neighboring videos in q-d graph to get it new representation $\mathbf{d}_d^{'}$

$$\mathbf{d}_d^{'} = \mathbf{d}_d \oplus \mathbf{d}_{d\_t} \oplus \mathbf{d}_{d\_v} \oplus \mathbf{d}_{dd}. \tag{6.13}$$

After we got the new embeddings for user, query and videos, we need to further refine the query representation by injecting more personalized information. Instead of directly

matching $\mathbf{q}_q^{'}$ and $\mathbf{d}_d^{'}$, we first combine $\mathbf{u}_u^{'}$ with $\mathbf{q}_q^{'}$ to get the personalized query embedding as following:

$$\mathbf{q}_q^p = f_u(\mathbf{u}_u^{'}) \oplus f_q(\mathbf{q}_q^{'}), \tag{6.14}$$

where f_u and f_q are MLPs and map $\mathbf{u}_u^{'}$ and $\mathbf{q}_q^{'}$ to the same vector space. Meanwhile, $\mathbf{d}_d^{'}$ will be also projected to the same space with $\mathbf{q}_q^p$ by an MLPs f_d

$$\mathbf{d}_d^p = f_d(\mathbf{d}_d^{'}). \tag{6.15}$$

The click probability of <u_u,q_q,d_d> is calculated as the inner product of $\mathbf{q}_q^p$ and $\mathbf{d}_d^p$,

$$\hat{y}_c = (\mathbf{q}_q^p)^T \mathbf{d}_d^p. \tag{6.16}$$

Besides, our model also considers the relevance between queries and videos and we calculate their correlation $\hat{y}_r$ by the inner product of $\mathbf{q}_q^{'}$ and $\mathbf{d}_d^{'}$

$$\hat{y}_r = (f_0(\mathbf{q}_q^{'}))^T f_1(\mathbf{d}_d^{'}), \tag{6.17}$$

where f_0, f_1 are MLPs to map $\mathbf{q}_q^{'}$ and $\mathbf{d}_d^{'}$ to the same embedding space. The final ranking score can be obtained as following

$$y = \hat{y}_c^{\beta} \hat{y}_r^{(1-\beta)}, \tag{6.18}$$

where $\beta \in (0,1)$ is a hyper-parameter. We use the final ranking score y to rank the candidate videos.

6.2.6 Model Training and Optimization

The training objective of our model consists of two terms. The first one is the binary cross-entropy loss

$$\mathcal{L}_1 = -\frac{1}{M}\sum_{j=1}^{M} o_j \times logp(\mathbf{d}_j|\mathbf{q}_q) + (1-o_j) \times log(1-p(\mathbf{d}_j|\mathbf{q}_q)), \tag{6.19}$$

where M denotes the number of training pairs and o_j represents binary click label for $\mathbf{d}_j$. The second term is the mean squared error

$$\mathcal{L}_2 = \frac{1}{M}\sum_{j=1}^{M}(y_j - \hat{y}_j), \tag{6.20}$$

where y_j and $\hat{y}_j$ are respectively the real and predicted relevance score. The final objective is

$$\mathcal{L} = \alpha\mathcal{L}_1 + (1-\alpha)\mathcal{L}_2, \tag{6.21}$$

where α is the hyper-parameters to control the balance.

6.3 Experiment

6.3.1 Datasets

To evaluate the effectiveness of our proposed framework, we collect the online search logs of a major commercial video search engine to construct a personalized search dataset, which could be a proper test bed to evaluate personalized search models. We collect a total of 21 days' search logs, which are split into 3 parts. Graphs are constructed based on the first part, i.e., search logs from the first 15 days. The constructed graphs have millions of nodes and tens of millions of edges. When constructing the two bipartite graphs, we only preserve five first-order and five second-order neighbors for each node. The next five days' logs are used to construct the training set and the logs from the last day are used as testing data. There are totally 1,289,314 unique users, 1,315,851 unique queries and 5,368,904 unique videos,

Table 6.1 Overall performance of all models (%) (mean ± 95% confidence interval over 20 runs)

. The best results are in **bold** and the best ones of other baselines are underlined.

		Click Task			
	Methods	AUC	nDCG@1	nDCG@3	nDCG@5
	DSSM	67.82 ± 0.19	20.8 ± 0.24	35.0 ± 0.16	40.6 ± 0.13
	NCF	72.48 ± 0.27	21.8 ± 0.27	37.4 ± 0.22	45.1 ± 0.19
	CDL	73.02 ± 0.09	22.7 ± 0.17	37.6 ± 0.16	44.5 ± 0.18
	Pclick	60.8	15.9	26.9	35.0
	NN-PVS	70.63 ± 0.12	23.3 ± 0.36	38.9 ± 0.27	46.2 ± 0.14
	RNN-PVS	68.33 ± 0.22	21.6 ± 0.19	37.0 ± 0.16	39.3 ± 0.18
GNN-PVS	*Mean*	76.93 ± 0.20	27.6 ± 0.28	44.0±0.15	51.6 ± 0.10
GNN-PVS	*Sum*	76.00 ± 0.09	25.3 ± 0.16	42.9±0.09	49.8 ± 0.04
GNN-PVS	*MaxPooling*	**78.87 ± 0.16**	27.8 ± 0.14	44.4 ± 0.10	51.7 ± 0.12
GNN-PVS	*Attention*	78.18 ± 0.12	**27.9 ± 0.22**	**45.0 ±0.21**	**51.9 ± 0.09**
MGNN-PVS	*Mean*	75.52 ± 0.21	27.2 ± 0.32	42.9 ± 0.14	50.6 ± 0.11
MGNN-PVS	*Sum*	74.67 ± 0.22	24.8 ± 0.17	40.5 ± 0.12	48.6 ± 0.09
MGNN-PVS	*MaxPooling*	77.03 ± 0.30	27.5 ± 0.36	43.4 ± 0.28	50.6 ± 0.25
MGNN-PVS	*Attention*	76.00 ± 0.11	26.8 ± 0.22	43.2 ± 0.14	50.7 ± 0.10
	Improvement%	+7.94%	+19.74%	+15.38%	+13.04%

where the training set consists of 10,802,573 samples, and testing set consists of 3,335,752 samples. For users, besides the user ID features, we also include users' meta information, such as age, education level, gender, profession and city, for better user representation.

For the click task, we formulate it as a binary classification task and collect the online user click feedback as the click label. Click means one and not click means zero. For the query-video relevance task, we formulate it as a regression task. The relevance labels are collected using an internal tool which could automatically tag the relevance label of a query-document pair. [5] The relevance labels range from zero to four, where four indicates the highest relevance level and zero means irrelevant.

[5]We could also get manual labels based on crowdsourcing platforms, such as Amazon Mechanical Turk.

Table 6.2 Overall performance of all models (%) (mean ± 95% confidence interval over 20 runs)

. The best results are in **bold** and the best ones of other baselines are underlined.

		Relevance Task		
	Methods	nDCG@1	nDCG@3	nDCG@5
	DSSM	67.0 ± 0.19	69.5 ± 0.17	72.4 ± 0.15
	NCF	66.8 ± 0.17	68.9 ± 0.13	71.7 ± 0.18
	CDL	68.6 ± 0.12	70.0 ± 0.09	72.3 ± 0.09
	Pclick	64.4	66.4	69.4
	NN-PVS	66.8 ± 0.14	67.6 ± 0.09	71.6 ± 0.09
	RNN-PVS	66.6 ± 0.17	67.3 ± 0.10	70.8 ± 0.07
GNN-PVS	*Mean*	70.5 ± 0.09	71.8 ± 0.07	74.0 ± 0.09
GNN-PVS	*Sum*	70.2 ± 0.13	72.4 ± 0.07	73.8 ± 0.09
GNN-PVS	*MaxPooling*	70.3 ± 0.34	71.8 ± 0.22	74.2 ± 0.20
GNN-PVS	*Attention*	70.6 ± 0.13	72.2 ± 0.09	74.9 ± 0.10
MGNN-PVS	*Mean*	72.2 ± 0.18	73.3 ± 0.11	**75.6 ± 0.12**
MGNN-PVS	*Sum*	71.7 ± 0.09	73.5 ± 0.11	74.1 ± 0.07
MGNN-PVS	*MaxPooling*	**72.3 ± 0.30**	72.9 ± 0.24	75.0 ± 0.20
MGNN-PVS	*Attention*	71.9 ± 0.20	**73.8 ± 0.10**	75.3 ± 0.11
	Improvement%	+5.12%	+5.43%	+4.42%

6.3.2 Baselines

We compare our model with two categories of baseline methods: classical information retrieval and semantic matching methods (NCF [77], DSSM [89], DNN) and three representative personalized search models: P-Click model (traditional statistic model) [49], two types of DL-based models: NN-PVS (neural ranking model with personalized information [88]) and RNN-PVS (RNN based sequential models to mining sequential information to learn users' search intents [4]). We evaluate above methods on the click task and the relevance task. In both tasks, we use AUC and a commonly used evaluation metric, i.e., nDCG@k (k=1, 3, 5), to evaluate these models and report results over 20 runs with random parameter initialization.

- **NCF** [77]. Neural Collaborative Filtering (NCF) is a neural network architecture to model latent features of users and items for top-K recommendation. In our scenarios, we feed it with the query IDs and video IDs instead of users and items.

- **DSSM** [89]. Deep Structured Semantic Model (DSSM) is proposed for web search semantic matching. We measure the similarities of query and videos' representations learned from their text information with BERT.

- **CDL**. Content-enhanced deep learning model (CDL) feed query ID embeddings concatenated with text embedding and video ID embeddings concatenated with text and video embeddings into the MLP layers.

- **P-Click model** [49]. P-Click model predicts the probability of clicking by counting the number of historical clicks. It focuses on the user's re-finding behavior.

- **NN-PVS** [88]. It is a neural ranking model with personalized information introduced in [88]. In this section, NN-PVS utilizes users' meta information to refine the query representation. We concatenate the user embedding learned from users' meta information with the query embedding and use the MLPs to map the concatenated vector to the same embedding space with the video for the final matching.

- **RNN-PVS**. This is an RNN based sequential model that learns users' profile with their history information [4]. We concatenate the user profile embedding with query embedding for the final matching.

- **GNN-PVS**. Our proposed model, where only click labels are used for training. We utilize four different aggregation functions: *Mean*, *Sum*, *MaxPooling* and *Attention*. [6]

- **MGNN-PVS**: It is a variant of GNN-PVS, where a multi-task learning framework is incorporated to simultaneously learn the click and relevance task. Likewise, we also utilize mean, sum, maxpooling and attention four different aggregation functions in our model.

[6]Considering accuracy, efficiency and stability, we use mean as our aggregation function if there is no specification.

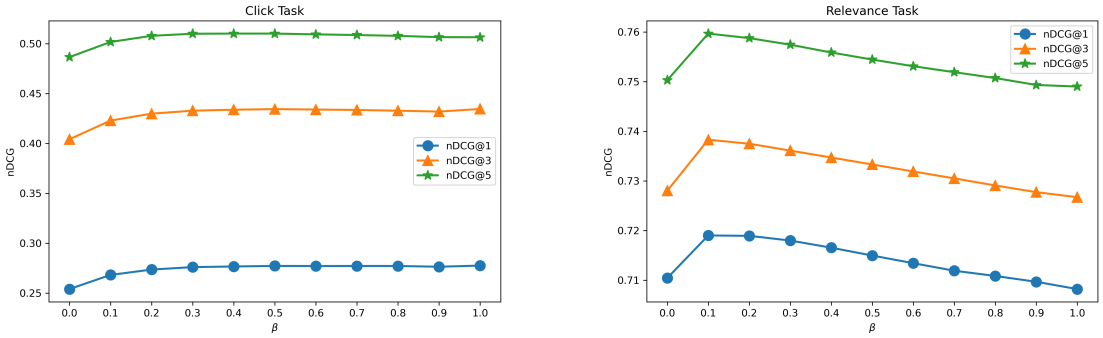
6.3.3 Hyperparameters Setting

The the dimension of text embedding is 128 and 256 for the video embedding. The other main hyper-parameters are set as: , the dimension of ID embedding $\in \{16, 32, 64\}$, the dimension of learned user profile $\in \{16, 32, 64\}$, the batch-size: 512, the dropout rate $\in \{0.2,0.4,0.6\}$, learning rate $\in \{0.01, 0.001, 0.0001\}$. These hyper-parameters are finally selected according to the performance on the validation dataset. We use the Adam optimizer [102] and apply the early stopping strategy based on the validation accuracy to avoid overfitting. Considering both accuracy and efficiency, we choose the mean aggregation function in our framework. α is set as 1 for GNN-PVS which only learns the click label and set as 0.5 for MGNN-PVS to jointly model click and relevance.

6.3.4 Performance for Click and Relevance Tasks.

The experimental results are summarized in Table 6.2. Our proposed GNN-based models (GNN-PVS and MGNN-PVS) significantly and consistently outperform all other baselines on both click and relevance tasks. Compared with the best baselines, our model outperforms the best baselines by 7.94%, 9.74%, 15.38%, 13.04% for the click task and 5.12%, 5.43%, 4.42% for the relevance task, which illustrates that utilizing graph information greatly benefits the representation learning for users, queries and videos. Compared with GNN-based models, MGNN-based models get better performance on the relevance task, but not the click task. This is because MGNN-based models are trained with both click and relevance labels, while GNN-based models are trained only with the click labels.

Besides, non-personalized baselines (DSSM, DCF, CDL) usually perform better than personalized search models (Pclick, NN-PVS, RNN-PVS) on the relevance task, but not the click task. The possible reason is that incorporating user's information benefits the click task, but may introduce some noisy information, such as users clicked the video only because of attractiveness, but it may be not relevant to their issued queries. This also indicates the necessity of training the model with both click and relevance labels.



(a) nDCG of Click Task (b) nDCG of Relevance Task

Fig. 6.5 β's influence to the prediction results.

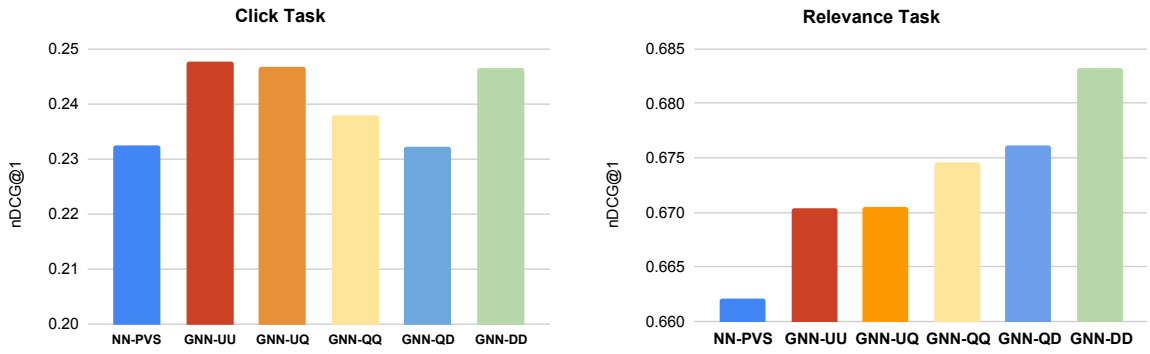
The P-Click model, the traditional statistical method, gets poor results, while other deep learning based models have shown strong generalization ability by capturing the implicit similarities among users, queries and documents. NN-PVS gets better results than RNN-PVS. One possible reason is that RNN is more suitable for sequential data, but there is no timestamp information for users' history information in our scenario. Compared with RNN-PVS, NN-PVS is a more general and easily applied PSM.

6.3.5 Trade-off between Two Tasks.

For MGNN-PVS, a key hyper-parameter is the trade-off parameter β. We would like to explore the influence of β to the performance on the click task and relevance task by varying the value of β from zero to one. Results are reported in Fig. 6.5. We found that MGNN-PVS achieves better performance on the relevance task when β is small. When the value of β increases, the performance of the click task increases while the performance of the relevance task decreases. This observation matches our intuition since β controls the relative impacts of click prediction score and relevance prediction on the final ranking score.

6.3.6 Effectiveness of Graph Information.

To validate the effectiveness of graph information, we add each component of the graph to examine its relative importance as shown in Fig. 6.6. For example, GNN-UU means



(a) nDCG@1 of click task

(b) nDCG@1 of relevance task

Fig. 6.6 Effectiveness of each graph module: (a) and (b) show the nDCG@1 of click task and relevance task. NN-PVS is the baseline, only considering the user's meta information. GNN-AB means only A-B graph information is used in the training process, such as GNN-UU means only the user's neighboring users (user-user graph information) are used in the learning process.

our model only leverages neighboring users' information (user-user) from the user-query graph, without considering other components of the two graphs. Several observations can be made from Fig. 6.6. First, adding user-user and user-query information has more improvements than adding query-query, query-documents information for the click task. When uses' history information is sparse, they may not be well represented. Hence, adding their neighboring users and queries may assist our model to learn better representation. Second, the improvement brought by second-order neighboring videos in the query-document graph outweighs that brought by other graph information for the relevance task. One possible reason is that videos can learn more accurate embeddings with considering their neighboring videos. Finally, compared with other graph information, the improvement brought by query-query and query-document graph information is not that obvious. We checked our dataset and found that the connection is very sparse for query in the query-document graph, and this may result that adding query-query and query-document graph information does not perform as good as adding user-user or video-video information.

6.3.7 Case Study.

To get deep insights on how the graph information assists the personalized search task. We choose two triples <*user*, *query*, *video*> with the same query *Shepherd of the Cocoa Sea* (*SCS*) and video *Beautiful street performance of the Shepherd of the Cocoa Sea*, but different users with the ground truth labels zero for *user47* (no-click) and one for *user75* (click). We compare the click results of NN-PVS and our model. NN-PVS predicts both *user47* and *user75* click the video when issuing this query *SCS*, because the query and video title are very relevant, containing both *Shepherd of the Cocoa Sea*. But our model predicts zero for *user47* and one for *user75*.

From the user-query graph, we find three related first-hop neighbors of *user47*: *the original singer of SCS*, *the complete and original SCS*, *Wang CSC* [7], which illustrates that *user47* prefers the original song *CSC* much. Beyond the text information of the query and video, our model can utilizes these information and discover people's real interests, but NN-PVS can not. Besides we also observe that the query *Cocoa Shepherd*, a wrong expression of *SCS*, has three second-hop neighbors (similar queries): *SCS*, *SCS orignal*, *Song of SCS* in the query-document graph, which contains more related and accurate expressions for the query *SCS*. Aggregating these information can help the model learn a more accurate and robust representation of queries, even with wrong expressions.

6.4 Summary

In this chapter, we proposed to utilize the user-query and query-video (click) graphs to assist the personalized video search task. While the graphs are heterogeneous and contain millions of nodes and edges, which are quite challenging to deal with. To overcome this problem (Q3), we apply the hop-aware projection strategy as in HHR-GNN (Chapter 5) to capture a given node's first-order (i.e., user-query) and second-order (i.e., user-user) interactions in the graphs, also present an efficient, localized convolution by sampling fixed-size neighbors from graphs. Besides, considering the real data contains much noisy click information that

[7] Wang is the original singer of CSC

users' click signals may indicate attractiveness but not necessarily indicate relevance, we jointly model the user's click behaviour and the relevance between queries and videos in our algorithm. Experimental results on the real-world dataset showed that our proposed model significantly outperforms state-of-the-art PSMs on both click task and relevance task, which illustrates the effectiveness of our proposed framework.

Chapter 7

KG-aware Cross-Domain Recommendation

7.1 Introduction

Cross-domain recommendation (CDR) [54] is a promising solution to the data sparsity problem in recommender systems. Conventional single-target CDR models leverage information from a richer (source) domain to improve the recommendation performance in a sparser (target) domain [85, 17]. For performance improvement in both domains, recent dual-target CDR models [131, 250, 114] enable bidirectional transfer across domains with dual-learning mechanism [241, 76].

Despite encouraging results from existing CDR models, several key issues remain [251]. Firstly, current models, including the dual-target ones, can not simultaneously improve the performance in both source and target domains due to negative transfer [147]. In general, the knowledge learned from the sparser domain is less accurate than that learned from the richer domain. Thus, the recommendation performance in the richer domain tends to decline if the transfer direction is simply inverted. The limited information, especially in the sparse domain, is one of the bottlenecks. Secondly, current CDR models mainly use common users [131, 250] or mapping functions [114] to build connections between domains. In real-life scenarios, relationships between items within or across domains can characterize item-wise

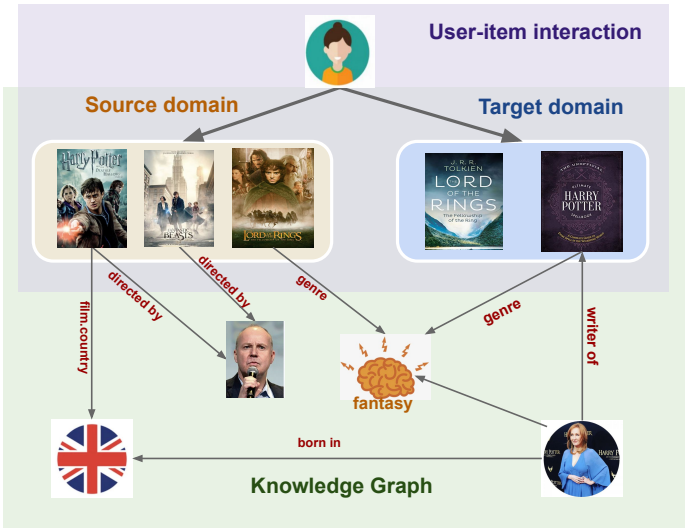


Fig. 7.1 Knowledge graph is a natural bridge that connects items from different domains. For example, "*Lord of the Ring*" in movies can get connected with "*Harry Potter*" in books via related genre *Fantasy*. Such inter-domain knowledge can reveal similar semantic relations among items from different domains to further improve cross-domain recommendation. We construct a new dataset and propose a new model to achieve this goal.

semantic relatedness to help understand user-item interaction patterns [203]. SemStim [78] exploits the semantic links found in a knowledge graph (e.g. DBpedia), to connect domains and thus generate recommendations for the target domain. However, the knowledge graph are usually very sparse and noisy, which may not an optimal choice to use the found products directly in the KG as the recommendations. Besides, SemStim is a single-target CDR approach that only focus on how to leverage the source domain to help improve the recommendation accuracy on the target one, but not vice versa. How to leverage the KG better to improve both the source and target domains' recommendation is a key issue to solve.

In this chapter, we aim to address this gap by leveraging knowledge graph (KG), a natural bridge for items from different domains [204]. KG can benefit the CDR task in multiple ways [200]. First, rich and explicit connections among items in a KG can help improve the recommendation performance in each domain, particularly the sparser domain. As shown in Fig 7.1, a user who has watched "*Harry Potter and the Deathly Hallows*" is very likely to have interest in the original novel, which can be recommended with the assistance of cross-domain knowledge. Second, domains often share some domain-general information.

For example, genre can characterize both book and movie domains. "*Lord of the Ring*" (from movies), "*Harry Potter*" (from books), "J. K. Rowling" (from KG), and other entities from different hops of neighbors can be closely connected in KG via the related genre *Fantasy*. KG provides a natural way to leverage such inter-domain knowledge that can help models understand target items by associating rich semantic relatedness among items, explore their latent connections, and further improve recommendation performance.

To build KG-aware CDR, three unique technical challenges arise. (1) Though several datasets exist for KG-aware single-domain recommendation, no publicly-available dataset exists for KG-aware CDR. (2) To improve CDR, item (entity) embeddings (representations) learned from KG should contain both domain-specific and domain-general information, which typically comes from different hops of neighbors in KG. The second challenge is to model both adjacent and higher-order relations in the item representation learning process. (3) Item embeddings learned from KG and those from the user-item interaction matrix should be closely related, e.g., highly correlated, so that cross-domain relationships can be effectively established. This is the third challenge.

To address the challenges above, we construct a new dataset for KG-aware CDR and propose a novel KG-aware Neural Collective Matrix Factorization (KG-aware NeuCMF) model. More specifically, our contributions are:

- We construct a new dataset named *Amazon product Knowledge Graph for CDR* (*AmazonKG4CDR)* using a subset (movie, book, and music) of the Amazon Review Data (2018) [140] and the Freebase KG [33, 18]. To the best of our knowledge, this is the first dataset that links KG information for CDR.

- We propose a two-step KG-aware NeuCMF framework for KG-aware CDR. We train a *shared* autoencoder using a relational graph convolutional network (RGCN) on KG following a contrastive learning-style [104, 161]. GCN-based encoders learn a node's embedding by aggregating information from its neighbors via non-linear transformation and aggregation [103]. Long-range node dependencies can be captured by stacking multiple GCN layers to propagate information for multiple hops [216]. This enables capturing both domain-specific and domain-general information from different hops

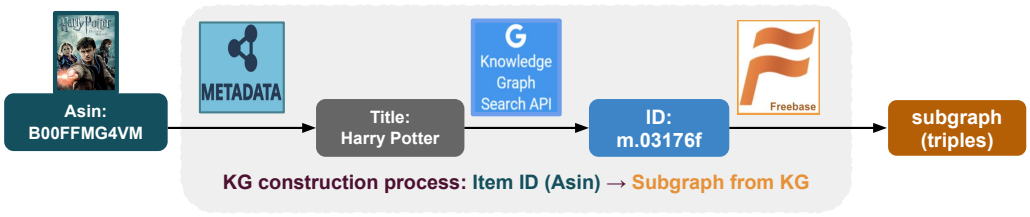


Fig. 7.2 KG construction for Amazon products.

of neighbors in KG. To establish cross-domain relationships, the embeddings learned from KG should be highly coherent with those from the user-item interaction matrix. Therefore, we incorporate the mutual information (MI) estimation [15] into the neural collective matrix factorization (NeuCMF) framework to jointly learn the two cross-domain rating matrices by sharing the user embeddings. This mechanism allows our model to preserve both user-item interaction and KG information across items.

- Finally, we conduct extensive experiments on our newly constructed datasets and demonstrate that our model significantly outperforms the best-performing baselines.

7.2 KG Construction for CDR

To develop a knowledge-aware CDR system, a key issue is how to obtain rich and structured knowledge information for items. Existing research works use side information from the original recommender system, such as tags and reviews. We argue that the KG information will provide additional useful information to the CDR task, since the intra-domain relationship among items can be captu. In this chapter, we present *AmazonKG4CDR V1.0*, a new dataset linking KG information for CDR, which can be useful for researchers in the related areas to explore possible approaches with the rich KG information.

We use the widely used dataset, Amazon Review Data (2018) [140], covering various domains, from which we select a subset that includes two domain pairs: movie-music, movie-book, which are being linked together through a common user ID identifying the same user. On the KG side, we use the well-known KG: Freebase [18]. It stores facts by triples

of the form $< head >< relation >< tail >$. Since Freebase shut down its services, we use its latest public version. We map items into Freebase entities via title matching if there is a mapping available. Fig.7.2 shows the whole linkage process. Since we only have item Asins (IDs of Amazon products), we need to get items' titles from the Amazon Review metadata first[1]. These titles are later used to get KG entity IDs from *The Knowledge Graph Search API*, which are used to extract the graph information from Freebase. We take triplets that involve two-hop neighbor entities of items into consideration.

During the linkage process, we have dealt with several problems that will affect the quality of the extract knowledge graph. First, the correctness of the extracted KG entity IDs should be ensured. For example, a query is *"Harry Potter"* (a book name), returned results can be both movies and books. So, we filter returned results by their type and name to ensure extracted IDs are correct. To ensure the KG quality, we preprocess the extracted KG by filtering out infrequent entities (e.g., lower than 10 in both datasets) and retaining the relations appearing in at least 100 triplets.

7.3 Methodology

In this section, we present the technical details of our proposed CDR model, KG-aware Neural CMF (KG-NeuCMF) that aims to improve the performance of CDR by leveraging the KG. We first formulate the task, then present our proposed framework KG-NeuCMF that aims to improve the performance of CDR by leveraging the KG. Fig.7.3 shows the overview of the proposed framework. In the first stage, we propose to learn KG-level representations by exploiting a multi-layer RGCN [161] through the encode-decode paradigm by minimizing the reconstruction loss that follows a contrastive learning-style convention [104]. This step aims to learn item embeddings containing both domain-specific and domain-general information from different hops of neighbors in KG. In the second-stage, we learn item and user embeddings by borrowing ideas from the CMF framework [170] and neural CF (NCF) [77]. Instead of jointly factorizing the two user-item interaction matrices directly as in CMF,

[1]https://nijianmo.github.io/amazon/index.html

we propose to utilize neural networks to jointly learn the two matrices by sharing user latent representations. Finally, item representations learned from KG and user-item interaction matrix should be highly correlated. To quantify such correlation, we also exploit to maximize MI [15] between the two types of representations.

7.3.1 Problem Definition

In this chapter, we study the problem of KG-aware CDR. Formally, we are given two domains, a source domain $\mathcal{S}$ (e.g., movie recommendation) and a target domain $\mathcal{T}$ (e.g., book recommendation) that can be represented as two user-item interaction matrices $\mathbf{R}_{\mathcal{S}}$ and $\mathbf{R}_{\mathcal{T}}$, where $r_{ui} = 1$ indicates that user u engages with item i, otherwise $r_{ui} = 0$. In real online shopping platforms (e.g., Amazon), users in domain $\mathcal{S}$ and domain $\mathcal{T}$ often overlap, meaning that they have purchased items in both domains. The set of users in both domains are shared, denoted by $\mathcal{U}$ (of size $m = |\mathcal{U}|$). In our setting, there is no overlap of items between two domains and each item only belongs to one single domain. Denote the set of items in $\mathcal{S}$ and $\mathcal{T}$ by $\mathcal{I}_{\mathcal{S}}$ and $\mathcal{I}_{\mathcal{T}}$ with size $n_{\mathcal{S}} = |\mathcal{I}_{\mathcal{S}}|$) and $n_{\mathcal{T}} = |\mathcal{I}_{\mathcal{T}}|$ respectively. Additionally, we also have a knowledge graph $\mathcal{G}$, a multi-relational graph, containing rich facts about items. Each fact in the KG is represented as a triple (head entity,relation,tail entity) ((h, r, t)), also called fact [204]. The KG can represent large-scale information from multiple domains [52]. In recommendation scenarios, an item in the user-item interaction matrix corresponds to an entity in KG.

Given $\mathbf{R}_{\mathcal{S}}$ and $\mathbf{R}_{\mathcal{T}}$ as well as the knowledge graph $\mathcal{G}$, we aim to predict whether user u will engage with item i with which the user has no interaction before. Our goal is to learn a prediction function $\hat{y}_{ui} = f\ (u, i \mid \Theta, \mathbf{R}_{\mathcal{S}}, \mathbf{R}_{\mathcal{T}}, \mathcal{G})$, where $\hat{y}_{ui}$ denotes the probability (or the rating score) that user u will engage with item i and Θ denotes the model parameters of function f.

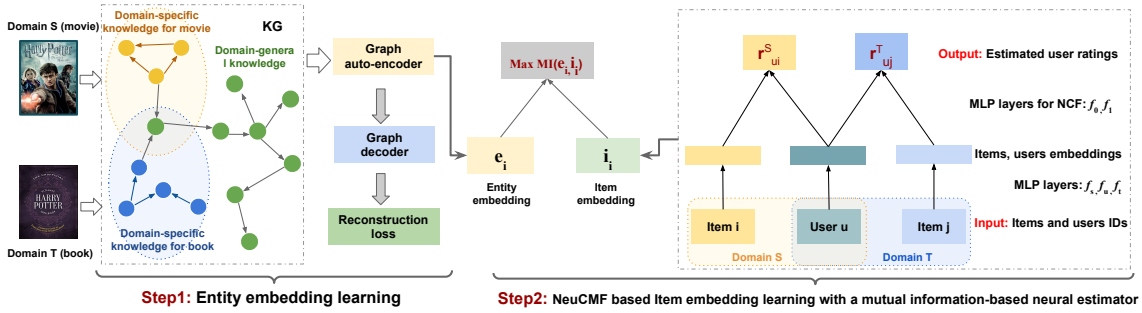


Fig. 7.3 The framework of our model: KG-aware NeuCMF. It learns item representations from both KG (left) and user-item interaction matrices (right). Entity (item) representations learned from KG contain both domain-specific and domain-general information by utilizing graph autoencoding strategy, which can help assist the CDR task. Item embeddings are learned by a neural CMF model. To ensure the two types of embeddings are highly correlated, we maximize their MI by the neural mutual information estimator (middle).

7.3.2 Entity Embedding

To utilize KG in our task, we first need to learn entity representations. We do this by training a graph autoencoder model in the unsupervised fashion and learn representations in an encode-decode paradigm [104, 161]. We employ RGCN [161] as our encoder that learns an entity embedding by aggregating information from its adjacent neighbors via non-linear transformation and aggregation dependent on the connecting relation, which can be denoted as

$$f_{en}(\mathbf{e}_i^{(l)}, \mathbf{e}_j^{(l)}) = \sigma(\mathbf{W}_0^{(l)}\mathbf{e}_i^{(l)} + \sum_{r\in\mathcal{R}}\sum_{j\in\mathcal{N}_i^r}\frac{1}{c_{ij}}\mathbf{W}_r^{(l)}\mathbf{e}_j^{(l)}), \tag{7.1}$$

where $\mathbf{e}_i^{(l)}\mathbf{e}_i^{(l)}$ are the hidden state of node i and node j in the $l-$th layer of the encoder, σ is an activation function such as ReLU, $\mathbf{W}_0^{(l)}$, $\mathbf{W}_r^{(l)}$ are (learnable parameters) relation-specific transformation mapping matrix depending on the type of edge, c_{ij} is problem-specific normalization constant that can either be learned or chosen in advance, and $\mathcal{N}_i^r$ denotes the set of neighbors of node i under relation $r \in \mathcal{R}$. Through this operation, the local proximity structure and related semantic information can be successfully captured and stored in the new representation of each entity. Long-range node dependencies can be captured by stacking

multiple graph encoder layers and this mechanism ensures that distinct domains can be connected via the information propagation.

The decoder can be any scoring function of KG embedding methods [204] that are used to measure the plausibility of each fact (h,r,t). Following [161], we use DisMult [217] factorization as the scoring function, which is well known for its simplicity and efficiency and a triple (h,r,t) is scored as

$$f_{de}(\mathbf{e}_h,\mathbf{r},\mathbf{e}_t) = \mathbf{e}_h\mathbf{R}_r\mathbf{e}_t, \tag{7.2}$$

where $\mathbf{e}_h,\mathbf{e}_t \in \mathbb{R}^d$ are encoded features vector for entity h and t, and each relation $\mathbf{r}$ is associated with a diagonal matrix $\mathbf{R}_r \in \mathbb{R}^{d\times d}$.

We train the encoder and decoder with negative sampling. We construct an equal number of negative samples by randomly replacing the head entity or tail entity of each positive sample and the overall set of samples are denoted by $\mathcal{M}$. Then we minimize the cross-entropy loss of positive and negative node pairs

$$\mathcal{L} = \sum_{(\mathbf{e}_h,\mathbf{r},\mathbf{e}_t,y)\in\mathcal{M}} (ylogf_{de}(\mathbf{e}_h,\mathbf{r},\mathbf{e}_t))) + (1-y)log(1-f_{de}(\mathbf{e}_h,\mathbf{r},\mathbf{e}_t)). \tag{7.3}$$

7.3.3 NeuCMF Module

Typically the user-item interaction matrices are highly sparse and it is beneficial to learn them simultaneously [170]. Collective matrix factorization (CMF) jointly factorizes two matrices by sharing the user latent factors. Motivated by neural CF (NCF) [77], we propose to utilize neural networks to jointly learn the two matrices by sharing user latent representations as shown in Fig. 7.3. The predicted scores in two domains are

$$r_{ui}^{\mathcal{S}} = f_0(f_u(\mathbf{u}_u), f_s(\mathbf{i}_i^{\mathcal{S}}), \tag{7.4}$$

$$r_{uj}^{\mathcal{T}} = f_1(f_u(\mathbf{u}_u)), f_t(\mathbf{i}_j^{\mathcal{T}})), \tag{7.5}$$

where $\mathbf{u}_u$, $\mathbf{i}_i^{\mathcal{S}}$ and $\mathbf{i}_j^{\mathcal{T}}$ are represented one-hot vectors of users, items from domain $\mathcal{S}$ and domain $\mathcal{T}$ respectively, where only the element corresponding to that index is 1 and all others are 0. f_u, f_s and f_t can be multi-layer perceptron (MLP) that project the sparse representation to dense vectors. The obtained embeddings are then feed into two separate multi-layer neural architectures to map the latent vectors to predict scores $r_{us}^{\mathcal{S}}$, $r_{ut}^{\mathcal{T}}$ for the two domains. Given $\mathbf{R}_{\mathcal{S}}$ and $\mathbf{R}_{\mathcal{T}}$, we minimize the two reconstruction losses $\mathcal{L}_{\mathcal{S}}$ and $\mathcal{L}_{\mathcal{T}}$ with the predicted scores.

The NeuCMF module connects two domains only by the common users, and fails to capture the relations among items. The item embedding learned from KG can capture both domain-specific and domain-general knowledge, thus will be effective for both single-domain and cross-domain recommendation. Intuitively, the learned item embedding from user-item interaction matrices should be highly correlated to the KG-level embeddings. Therefore, this motivates us to exploit to maximize MI [15] between the two types of representations to guarantee their highly correlated relationship. We design our neural mutual information estimator based on a discriminator $\mathcal{D}(x,y)$ for their pairwise relationships, to provide probability scores for sampled pairs. To be specific, we generate positive samples as ($\mathbf{e}_i$,$\mathbf{i}_i$) ($\mathbf{i}$ can come from domain $\mathcal{S}$ and domain $\mathcal{T}$, half-half) and negative samples are generated by associating sampled items with fake embeddings based on shuffling strategy [192]. We define the loss function as:

$$\mathcal{L}_{mul} = -\frac{1}{N_{pos}+N_{neg}}\left(\sum_{i=1}^{N_{pos}} \mu(\mathbf{i}_i,\mathbf{e}_i) log\sigma(\mathbf{i}_i,\mathbf{e}_i) + \sum_{i=1}^{N_{neg}} \mu(\tilde{\mathbf{i}}_i,\mathbf{e}_i) log\sigma(\tilde{\mathbf{i}}_i,\mathbf{e}_i)\right), \tag{7.6}$$

where N_{pos}, N_{neg} denotes the number of positive and negative samples, $\mu(\cdot)$ is an indicator function, e.g., $\sum_{i=1}^{N_{pos}} \mu(\mathbf{i}_i,\mathbf{e}_i) = 1$ and $\sum_{i=1}^{N_{neg}} \mu(\tilde{\mathbf{i}}_i,\mathbf{e}_i) = 1$ corresponds to positive and negative pair samples. We aim to minimize $\mathcal{L}_{mul}$, which is equivalent to maximize the mutual information, to jointly preserve the KG-level and user-item interaction information.

7.3.4 Model Training and Optimization

The final loss includes: the loss ($\mathcal{L}_{\mathcal{S}}$) of source and loss ($\mathcal{L}_{\mathcal{T}}$) of target recommendation with the mutual information maximization loss $\mathcal{L}_{mul}$. The objective is to minimize the overall loss

Table 7.1 Statistics of the dataset.

	Domain: Music-Movie		Domain: Book-Movie	
	Music	Movie	Book	Movie
Users	4,196	4,196	3,977	3,977
Items	7,412	10,919	11,372	8,118
Interactions	21,986	49,027	22,214	29,245
Entities	85,612	387,178	258,999	990,141
Relations	155	340	127	295
Triples	288,731	610,314	522,814	1,787,190

$\mathcal{L}$ as follows:

$$\mathcal{L} = \mathcal{L}_{\mathcal{S}}(\Theta_{\mathcal{S}}) + \mathcal{L}_{\mathcal{T}}(\Theta_{\mathcal{T}}) + \mathcal{L}_{mul}(\Theta_{mul}) + \lambda \|\Theta\|, \tag{7.7}$$

where $\Theta = \Theta_{\mathcal{S}} \cup \Theta_{\mathcal{T}} \cup \Theta_{\mathcal{L}_{mul}}$. Note that $\Theta_{\mathcal{S}}$ and $\Theta_{\mathcal{T}}$ share user embeddings. The objective function can be optimized by stochastic gradient descent (SGD) and its variants like adaptive moment method (Adam) [102].

7.4 Experiment

7.4.1 Datasets

We use the Amazon Review Data (2018) [140] that is widely used for product recommendation. It contains users' rate (ranging from 1 to 5) for product from various domains. We select a subset that includes two domain pairs: movie-music(MM), movie-book(MB), which are being linked together through a common user ID identifying the same user. We construct the knowledge graph for each item by utilizing Freebase and the basic statistics details are presented in Table 7.1. The recommendation task can be formulate as the regression (rating) or the binary classification (recommend or not) tasks. Following [155], we evaluate the recommendation performance based MAE, F1_score (Threshold of positive rating is 4) for the regression and classification performance, respectively.

7.4.2 Baselines

To validate the performance of the proposed model, we compare the performance with five representative models, in which two single-domain RS models (MF, NCF) and three CDR models (CMF, CoNet, DDTCDR) using the publicly released implementations.

- **MF** [106]. Matrix Factorization (MF) is a classic latent factors CF approach which learns the user and item factors via matrix factorization in each domain separately.
- **NCF** [77]. Neural Collaborative Filtering (NCF) is a neural network architecture to model latent features of users and items using CF method. The NCF models are trained separately for each domain without transferring any information.
- **CMF** [170]. Collective Matrix Factorization (CMF) jointly factorizes matrices of each domains. In our scenarios, The shared user factors enable knowledge transfer between cross domains .
- **CoNet** [84]. Collaborative Cross Networks (CoNet) enables dual knowledge transfer across domains by introducing cross connections from one base network to another and vice versa.
- **DDTCDR** [114]. Deep Dual Transfer Cross Domain Recommendation (DDTCDR) learns latent orthogonal mappings across domains and provides cross domain recommendations by leveraging user preferences from all domains.

7.4.3 Hyperparameters Setting

In the KG-pretrain step, we utilize a two-layer RGCN as the encoder to obtain entity embeddings with hidden dimension 16. In the NeuCMF module, we apply one-layer neural networks to project the one-hot vectors of users, and items to low-dimensional embedding vectors and f_0 and f_1 are two one-layer neural networks to map the latent vectors to predict scores. Throughout the experiments, the embedding size is tuned in the range of [8,16,32] and we use the Adam optimizer [102] with learning rate 0.001, L2 regularization 0.0001. For

Table 7.2 Comparison of recommendation performance in Movie-Music (%) (mean ± 95% confidence interval over 100 runs). Best results: **bold**, second best ones: underlined.

Methods	Movie-Music (MM)			
	Movie		Music	
	MAE	F1_Score	MAE	F1_Score
MF [106]	20.94±0.480	74.97±0.804	23.79±0.314	72.57±0.118
NCF [77]	19.01±0.018	88.93±0.009	15.25±0.627	93.05±0.078
CMF [170]	20.23±0.372	89.09±0.069	11.66±0.064	92.45±0.068
CoNET [84]	18.22±0.068	88.68±0.139	13.96±0.070	92.05±0.078
DDTCDR [250]	20.69±0.066	74.84±0.262	15.82±0.137	89.05±0.425
Ours	**14.23±0.176**	**90.69±0.043**	**9.89±0.063**	**94.45±0.043**
Improvement (%)	21.28 %	1.80 %	15.18 %	1.50%

Table 7.3 Comparison of recommendation performance in Movie-Book(%) (mean ± 95% confidence interval over 100 runs). Best results: **bold**, second best ones: underlined.

Methods	Movie-Book (MB)			
	Movie		Book	
	MAE	F1_Score	MAE	F1_Score
MF [106]	24.17±0.239	73.64±0.139	23.83±0.225	69.01±0.421
NCF [77]	18.80±0.099	89.08±0.009	18.86±0.102	89.35±0.012
CMF [170]	14.53±0.296	89.32±0.001	13.22±0.147	89.07±0.045
CoNET [84]	17.46±0.119	89.59±0.276	17.18±0.106	89.22±0.145
DDTCDR [250]	20.17±0.104	82.60±0.496	17.15±0.104	90.06±0.066
Ours	**13.17±0.025**	**90.60±0.060**	**13.01±0.021**	**90.80±0.0276**
Improvement (%)	9.36 %	1.12 %	1.58 %	0.57%

each dataset, the ratio of training, evaluation, and test set is 6 : 2 : 2 [201]. We employ the early stopping strategy based on the validation accuracy with a window size of 10 (we will stop training if the validation loss does not decrease for 10 consecutive epochs) and train 200 epochs at most. We report results over 100 runs with random weight matrix initialization. For a fair comparison, we set the same hyperparameters of the baselines as our model.

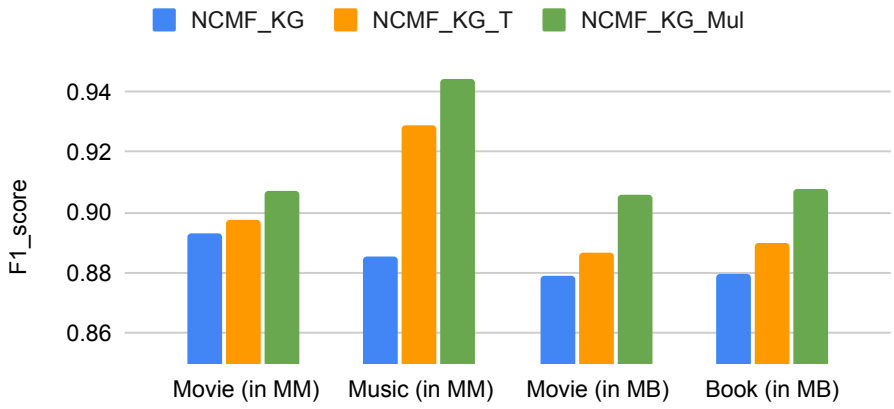


Fig. 7.4 Different ways to incorporate KG information for CDR.

7.4.4 Performance for CDR

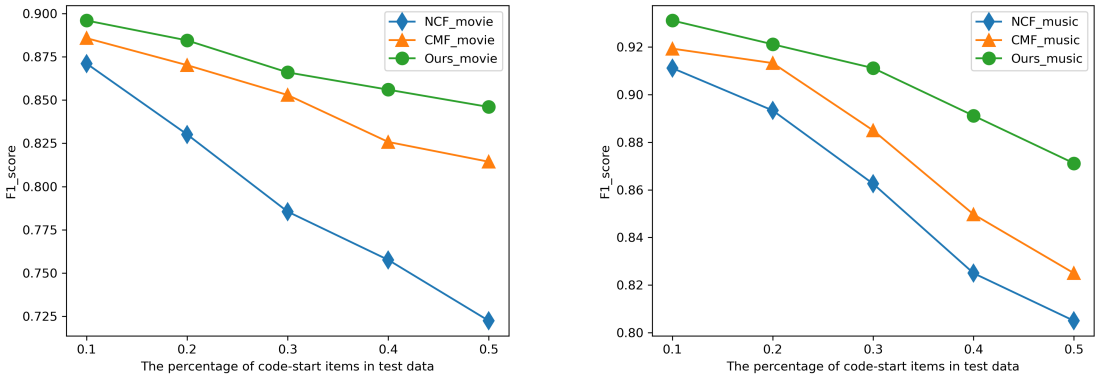
We have conducted experiments on two cross domain tasks, movie-music (MM) and movie-book (MB), and the corresponding results of our model and baselines are shown in Table 7.2 and Table 7.3. We can see that our proposed model can consistently obtain the best performance across movie-music and movie-book recommendations in terms of MAE and F1_score. In particular, our model improves over the strongest baselines *w.r.t*. MAE by 21%, 15.18% in movie, music (Table 7.2) respectively, which justifies the effectiveness of our method in integrating items' KG information. If we compare between these two tasks, MM and MB, the improvement on music in MM is more remarkable compared to the performance in MB. Possible reasons are 1) the data is more sparse in the user-music interaction matrix, so leveraging KG information can greatly relieve the sparsity problem (we have verified this in the later experiments: Comparisions for cold-start item scenarios); 2) the extracted KG contains much useful information, especially for two closely related domains (movie and music both belong to multi-media datasets). Besides, CDR models (CMF,CoNet,DDTCDR) achieve better performance than SDR models (MF, NCF), indicating that utilizing extra information from other resources benefits the performance of recommendation.

7.4.5 Different Ways to Incorporate KG Information

We explore different ways to combine item embeddings learned from KG and user-item interaction matrices. NMF_KG takes KG-level embeddings as input, then incorporates them with item embeddings learned from user-item interaction matrices via an aggregation method, e.g., concatenation. NCMF_KG_T tries to refine item embeddings learned from KG with a one-layer MLP and concatenates with embeddings learned from the user-item interaction matrix. NCMF_KG_mul maximizes MI between the two types of representations to guarantee the highly correlated relationship. The results are shown in Fig. 7.4. Generally, refining the learned KG-level embeddings gets better performance than direct utilization. This is because in real-world KGs (e.g., Freebase) some noises are inevitably introduced in the process of automatically constructing large-scale KGs due to limited labour supervision [211, 95]. NCMF_KG_mul gets the best performance. The possible reason is that item embeddings jointly learn from the user-item rating matrix and entity embeddings from KG, which contain both domain-general and domain-specific knowledge and the neural mutual information estimator can ensure their correlation. Such design is more suitable for the cross-domain recommendation task.

7.4.6 Performance in Cold-Start Item Scenarios

KG is a natural bridge for items from different domains, which can further alleviate the item cold-start problem in RS. To validate this, we compare our methods with NCF, CMF under the code-start scenario. We set up the cold-start environment by sampling a subset of items for testing which are unseen in the training data. Results for cold-start items on movie-music datasets are shown in Fig. 7.5. NCF (the SDR model) is greatly influenced and gets the poorest performance, especially there are a large proportion new items. CMF (the CDR model) can leverage information from two domains, thus it can alleviate the cold-start problem in some extent. Our model goes further to learn representations for cold items from the KG, offering additional information beyond user-item interaction matrices.



(a) Movie (in MM (b) Music (in MM)

Fig. 7.5 Comparison of different models in cold-start items scenarios.

7.5 Summary

In this chapter, we utilized the knowledge graph for the cross-domain recommendation task. In order to learn both the domain-specific and domain-general knowledge from KG with graph autoencoding strategy (Q3), we apply the relation-specific projection as in HHR-GNN (Chapter 5) to learn item embeddings to capture both adjacent(domain-specific) and higher-order(domain-general) neighborhood information in the KG. Besides, we also constructed a new dataset *AmazonKG4CDR*, the first in the filed linking KG information for cross-domain recommendation. Moreover, we proposed a KG-aware NeuCMF model that unified item embeddings learned from user-item interaction matrices and KG with a neural collaborative filtering framework under a mutual information-based neural estimator. Through extensive experiments on real-world datasets, we demonstrated that KG-aware NeuCMF has achieved substantial gains over state-of-the-art baselines. For future work, we will build a larger dataset with more users, items, and domains, and explore the explainability of cross-domain recommendation.

Chapter 8

Conclusions and Future Directions

This thesis has explored local neighborhood information for graph representation learning. We summarise this thesis via the solved research questions as shown in Chapter 1 as below:

8.1 Conclusions

- **Feature-level attention of local neighborhood information**. Real-world graphs are noisy and adjacent nodes do not necessarily imply similarity. Some strategies have been proposed that allow for node-level attention of local neighborhood information, however, all individual features in a node feature vector are still treated equally. Each feature within a neighbor feature vector may play a different role for the central node's representation learning. To tackle this problem (Q1: node-level and feature-level attention of the local neighborhood information), we propose two new models: NFC-GCN (in Chapter 3) and LA-GCN (in Chapter 4). In Chapter 3, we proposed a new node-feature convolutional (NFC) layer to work on a fixed-size and ordered feature map that contains features from selected neighbors. Considering the irregular connectivity and lack of orderliness of graphs, we further extend NFC-GCN and proposed a more flexible and general framework: LA-GCN in Chapter 4, where there are no limitations for the size and order of the input data. Under this framework, we proposed a new aggregator function, *mask aggregator* that can learn a specific mask for each neighbors

before the neighborhood aggregation step. Both the NFC layer and mask-aggregator can assign different weights to different features in the learning process. Thus, NFC-GCN and LA-GCN successfully achieve node-level and feature-level attention of local neighborhood information in the learning process.

- **Hop-level attention of local neighborhood information**. For node representation learning, only the first-order neighborhood information may not be enough, which is caused by the sparsity or heterophily of graphs. Thus, some GNNs tried to leverage multi-scale neighborhood information to assist a given node's representation learning. Intuitively, neighbors from different hops show different importance for the central node's representation learning. However, existing methods usually aggregate the central node with different hops of neighbors directly and fail to model their differences in the learning process. To overcome this problem (Q2:hop-level attention of the local neighborhood), we proposed HHR-GNN in Chapter 5 where knowledge graph embedding techniques are introduced to learn the relationship between a given node and its different hops (types) of neighbors. The learned relation-score can be used to define a personalized receptive field for each node and a hop-aware aggregation to distinguish different hops or types of neighbors.

- **Exploring graph local information for application-specific tasks**. Due to the data sparsity (e.g., users' search history, products' rating information) in real industry, graphs, such as click graphs (CGs), knowledge graphs (KGs), can be used to alleviate this issue. For example, we can utilize the user relational graph to assist the user profile learning and knowledge graph to enrich the product's representation learning. However, industrial graphs are usually large-scale, sparse, and heterogeneous (containing different types of nodes and edges), which are quite challenging to deal with. How to apply GNNs to leverage the real-world graphs efficiently and effectively to assist some application-specific tasks is a key problem (Q3) we need to obstacle. In Chapter 6 and Chapter 7, we applied the sampling strategy and hop-specific transformation as in HHR-GNN, which enables us efficiently utilize the extra local neighborhood

information from graphs for a given node's representation learning. Then, the learned embedding from graphs can provide extra information of the target nodes (e.g., users, products), which benefits the personalized video search task and in Chapter 6 and the cross-domain recommendation task in Chapter 7 respectively.

8.2 Discussion

GNN-based research area is a young and promising research field. Our study has explored how to better leverage the graph local neighborhood information to learn better node representations. NFC-GCN in Chapter 3 and LAGCN in Chapter 4 focus on node-level attention and feature-level attention of the local neighborhood information, which can deal graphs with node feature better than conventional GNNs. The proposed NFC layer or mask-aggregator can be a plug-in module and integrated with other GNN extensions. Different from NFC-GCN and LA-GCN in Chapter 3 and Chapter 4, HHR-GNN 5 can explore higher-order neighborhood information and is suitable for both homogeneous and heterogeneous graphs. In Part II, we applies the proposed neighborhood aggregation strategies to the real industrial graphs for two application-specific tasks: personalized video search and cross-domain recommendation. Experimental results show that our proposed methods MGNN-PVS in Chapter 6 and KG-aware NeuCMF in Chapter 7 are both efficient and effective.

However, our models failed to deal with some important problems in this thesis.

- In this thesis, we only explore two types of graph local patterns (node features and different hops). Different types of local information, such as network motifs, ego networks, circles or tree-like structures usually demonstrate unique patterns, semantics of graphs [203, 202] and reveal more complex interactions within the data that go beyond pairwise interactions [45]. But, we have not yet explored in this thesis.

- All the proposed models are applied on static graphs, where nodes and edges are fixed and do not change over time. However, real networks are often dynamic. For instance, new users appear in the social networks, new transitions appear between two banks in

the financial transaction graphs [13, 100]. The proposed models in this thesis did not consider this.

- This thesis mainly focus on exploring the local information of a given node to assist general graph-related and application-specific tasks. Leveraging local neighborhood information to assist a given node's representation learning is a core strength, but also a major vulnerability of GNN [41, 253]. We only explore the robustness of the GNN model in Chapter 3, but other chapters did not explore the model's robustness.

8.3 Future Directions

While some fundamental problems of GNN have been address in this thesis, there are still many open problems to be considered. This section lists three research topics that worth further investigation.

- **Subgraph structures.** In this thesis, we only explore two types of graph local patterns (node features and different hops). Subgraphs, such as network motifs, ego networks, circles or tree-like structures, can be simple blocks and highly related to the functions of graphs in many domains, especially the biochemistry, ecology [7, 137, 230]. Hence, investigating the subgraph structure in the learning process is a crucial step for the graph-related tasks. However, these important subgraph structures have rarely been exploited by current GNNs in both algorithms and graph-related applications [133]. Therefore, extending current GNN architectures by exploring various subgraph structures could be an interesting and important direction for both algorithms and applications.

- **Dynamic GNNs.** Many real-world problems (e.g., financial transitions, social interactions) can be modelled as dynamic graphs where nodes and edges appear over time. Representing graphs as structures changing over time allows models to leverage not only structural but also temporal patterns [100]. Therefore, dynamic graphs are becoming an increasingly important topic of study. GNNs are primarily developed

for static graphs that do not change over time and some dynamic GNNs (D-GNNs) have been proposed to model dynamic graphs [149, 177, 94]. These models mainly focused on dynamic graph topology represented as a 3-dimensional cube. However, both the topology and node/edge attributes (features) of real graphs may vary with time and this can be considered another dimension in the dynamic graph (hypercube). Therefore, a future direction would be to extend current GNN models for dealing with both the dynamic graph structure and node/edge attributes, which will bring significant improvements to the expressive ability of existing D-GNNs.

- **Robust GNNs.** Because of the message passing between a given node and its neighbors in a graph, an attacker can change a single node's prediction without even changing any of its attributes or edges [35, 254]. This vulnerability has arisen tremendous concerns for adapting GNNs in safety-critical applications, such as financial systems, risk management. For example, in a credit scoring system, fraudsters can fake connections with several high-credit customers to evade the fraud detection models; spammers can easily create fake followers to increase the chance of fake news being recommended and spread [97]. What types of local neighborhood information and how to utilize this information to design robust GNN models would be an important future direction for both algorithms and applications.

References

[1] Abu-El-Haija, S., Perozzi, B., Kapoor, A., Harutyunyan, H., Alipourfard, N., Lerman, K., Steeg, G. V., and Galstyan, A. (2019). Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *ICML*.

[2] Aggarwal, C. C., Hinneburg, A., and Keim, D. A. (2001). On the surprising behavior of distance metrics in high dimensional space. In *International conference on database theory*, pages 420–434. Springer.

[3] Agichtein, E., Brill, E., and Dumais, S. (2019). Improving web search ranking by incorporating user behavior information. *SIGIR*.

[4] Ahmad, W. U., Chang, K.-W., and Wang, H. (2018). Multi-task learning for document ranking and query suggestion. In *ICLR*.

[5] Ahmed, A., Shervashidze, N., Narayanamurthy, S. M., Josifovski, V., and Smola, A. J. (2013). Distributed large-scale natural graph factorization. In *WWW*.

[6] Akbari, H., Yuan, L., Qian, R., Chuang, W.-H., Chang, S.-F., Cui, Y., and Gong, B. (2021). Vatt: Transformers for multimodal self-supervised learning from raw video, audio and text. *ArXiv*, abs/2104.11178.

[7] Alon, U. (2007). Network motifs: theory and experimental approaches. *Nature Reviews Genetics*, 8(6):450–461.

[8] Andrew, G., Arora, R., Bilmes, J. A., and Livescu, K. (2013). Deep canonical correlation analysis. In *ICML*.

[9] Arora, S. (2020). A survey on graph neural networks for knowledge graph completion. *arXiv preprint arXiv:2007.12374*.

[10] Aslam, J. A. and Frost, M. (2003). An information-theoretic measure for document similarity. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 449–450.

[11] Atwood, J. and Towsley, D. (2016). Diffusion-convolutional neural networks. In *NeurIPS*.

[12] Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. *ICLR*.

[13] Barros, C. D., Mendonça, M. R., Vieira, A. B., and Ziviani, A. (2021). A survey on embedding dynamic graphs. *ACM Computing Surveys (CSUR)*, 55(1):1–37.

[14] Beeferman, D. and Berger, A. (2000). Agglomerative clustering of a search engine query log. In *SIGKDD*.

[15] Belghazi, M. I., Baratin, A., Rajeshwar, S., Ozair, S., Bengio, Y., Courville, A., and Hjelm, D. (2018). Mutual information neural estimation. In *ICML*.

[16] Belkin, M. and Niyogi, P. (2001). Laplacian eigenmaps and spectral techniques for embedding and clustering. In *NeurIPS*.

[17] Berkovsky, S., Kuflik, T., and Ricci, F. (2007). Cross-domain mediation in collaborative filtering. In *UMAP*.

[18] Bollacker, K., Evans, C., Paritosh, P., Sturge, T., and Taylor, J. (2008). Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*.

[19] Bordes, A., Chopra, S., and Weston, J. (2014a). Question answering with subgraph embeddings. In *EMNLP*.

[20] Bordes, A., Glorot, X., Weston, J., and Bengio, Y. (2014b). A semantic matching energy function for learning with multi-relational data. *Machine Learning*, 94(2):233–259.

[21] Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., and Yakhnenko, O. (2013). Translating embeddings for modeling multi-relational data. In *NeurIPS*.

[22] Bordes, A., Weston, J., and Usunier, N. (2014c). Open question answering with weakly supervised embedding models. In *ECMLPKDD*.

[23] Bougouin, A., Boudin, F., and Daille, B. (2013). Topicrank: Graph-based topic ranking for keyphrase extraction. In *IJCNLP*.

[24] Brody, S., Alon, U., and Yahav, E. (2021). How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*.

[25] Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. (2014). Spectral networks and locally connected networks on graphs. In *ICLR*.

[26] Cao, S., Lu, W., and Xu, Q. (2015). Grarep: Learning graph representations with global structural information. In *CIKM*.

[27] Cao, Y., Long, M., Wang, J., Yang, Q., and Yu, P. S. (2016). Deep visual-semantic hashing for cross-modal retrieval. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[28] Carlson, A., Betteridge, J., Kisiel, B., Settles, B., Hruschka, E. R., and Mitchell, T. M. (2010). Toward an architecture for never-ending language learning. In *AAAI*.

[29] Carman, M. J., Crestani, F., Harvey, M., and Baillie, M. (2010). Towards query log based personalization using topic models. In *CIKM*.

[30] Carreira, J. and Zisserman, A. (2017). Quo vadis, action recognition? a new model and the kinetics dataset. In *CVPR*.

[31] Catherine, R. and Cohen, W. (2016). Personalized recommendations using knowledge graphs: A probabilistic logic programming approach. In *RecSys*.

[32] Cha, S.-H. (2007). Comprehensive survey on distance/similarity measures between probability density functions. *City*, 1(2):1.

[33] Chah, N. (2017). Freebase-triples: A methodology for processing the freebase data dumps. *arXiv preprint arXiv:1712.08707*.

[34] Chamberlain, B. P., Clough, J., and Deisenroth, M. P. (2017). Neural embeddings of graphs in hyperbolic space. *CoRR*.

[35] Chang, H., Rong, Y., Xu, T., Huang, W., Zhang, H., Cui, P., Zhu, W., and Huang, J. (2020). A restricted black-box adversarial framework towards attacking graph embedding models. In *AAAI*.

[36] Chen, J., Ma, T., and Xiao, C. (2018). Fastgcn: Fast learning with graph convolutional networks via importance sampling. In *ICLR*.

[37] Cong, W., Ramezani, M., and Mahdavi, M. (2021). On the importance of sampling in learning graph convolutional networks. *arXiv preprint arXiv:2103.02696*.

[38] Craswell, N. and Szummer, M. (2007). Random walks on the click graph. In *SIGIR*.

[39] Cucala, D. J. T., Grau, B. C., Kostylev, E. V., and Motik, B. (2022). Explainable GNN-based models over knowledge graphs. In *ICLR*.

[40] Cui, G., Zhou, J., Yang, C., and Liu, Z. (2020). Adaptive graph encoder for attributed graph embedding. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.

[41] Dai, H., Li, H., Tian, T., Huang, X., Wang, L., Zhu, J., and Song, L. (2018). Adversarial attack on graph structured data. In *ICML*.

[42] Defferrard, M., Bresson, X., and Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. In *NeurIPS*.

[43] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*.

[44] Dong, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmann, T., Sun, S., and Zhang, W. (2014). Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *SIGKDD*.

[45] Dong, X., Thanou, D., Rabbat, M., and Frossard, P. (2019). Learning graphs from data: A signal representation perspective. *IEEE Signal Processing Magazine*, 36(3):44–63.

[46] Dong, Y., Chawla, N. V., and Swami, A. (2017). metapath2vec: Scalable representation learning for heterogeneous networks. In *SIGKDD*.

[47] Dong, Y., Hu, Z., Wang, K., Sun, Y., and Tang, J. (2020). Heterogeneous network representation learning. In *IJCAI*.

[48] dos Santos, V. and Lifschitz, S. (2021). A semantic search approach for hyper relational knowledge graphs. *Anais Estendidos do XXXVI Simpósio Brasileiro de Banco de Dados (SBBD Estendido 2021)*.

[49] Dou, Z., Song, R., and Wen, J.-R. (2007). A large-scale evaluation and analysis of personalized search strategies. In *WWW*.

[50] Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. (2015). Convolutional networks on graphs for learning molecular fingerprints. In *NeurIPS*.

[51] Dzabraev, M., Kalashnikov, M., Komkov, S. A., and Petiushko, A. (2021). Mdmmt: Multidomain multimodal transformer for video retrieval. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 3349–3358.

[52] Ehrlinger, L. and Wöß, W. (2016). Towards a definition of knowledge graphs. *SEMANTiCS*.

[53] Fernández-Tobías, I. and Cantador, I. (2014). Exploiting social tags in matrix factorization models for cross-domain collaborative filtering. In *RecSys*.

[54] Fernández-Tobías, I., Cantador, I., Kaminskas, M., and Ricci, F. (2012). Cross-domain recommender systems: A survey of the state of the art. In *CERI*.

[55] Fu, X., Zhang, J., Meng, Z., and King, I. (2020). Magnn: Metapath aggregated graph neural network for heterogeneous graph embedding. In *WWW*.

[56] Gabeur, V., Sun, C., Alahari, K., and Schmid, C. (2020). Multi-modal transformer for video retrieval. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part IV 16*, pages 214–229. Springer.

[57] Gao, H. and Huang, H. (2018). Deep attributed network embedding. In *IJCAI*.

[58] Gao, H., Wang, Z., and Ji, S. (2018). Large-scale learnable graph convolutional networks. In *SIGKDD*.

[59] Gao, Z., Liu, J., Chen, S., Chang, D., Zhang, H., and Yuan, J. (2021). Clip2tv: An empirical study on transformer-based methods for video-text retrieval. *ArXiv*, abs/2111.05610.

[60] Ge, S., Dou, Z., Jiang, Z., Nie, J.-Y., and Wen, J.-R. (2018). Personalizing search results using hierarchical rnn with query-aware attention. In *CIKM*.

[61] Ge, Y., Ma, J., Zhang, L., and Lu, H. (2020). Unifying homophily and heterophily network transformation via motifs. *arXiv preprint arXiv:2012.11400*.

[62] Ge, Y., Peng, P., and Lu, H. (2021). Mixed-order spectral clustering for complex networks. *Pattern Recognition*, 117:107964.

[63] Gehring, J., Auli, M., Grangier, D., and Dauphin, Y. (2017). A convolutional encoder model for neural machine translation. In *ACL-IJCNLP*.

[64] Getoor, L., Friedman, N., Koller, D., Pfeffer, A., and Taskar, B. (2007). Probabilistic relational models. *Introduction to statistical relational learning*, 8.

[65] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *ICML*.

[66] Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*.

[67] Gong, Y., Ke, Q., Isard, M., and Lazebnik, S. (2013). A multi-view embedding space for modeling internet images, tags, and their semantics. *International Journal of Computer Vision*, 106:210–233.

[68] Gori, M., Monfardini, G., and Scarselli, F. (2005). A new model for learning in graph domains. In *IJCNN*.

[69] Goyal, P. and Ferrara, E. (2018). Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94.

[70] Grover, A. and Leskovec, J. (2016). node2vec: Scalable feature learning for networks. In *SIGKDD*.

[71] Guo, Q., Zhuang, F., Qin, C., Zhu, H., Xie, X., Xiong, H., and He, Q. (2020). A survey on knowledge graph-based recommender systems. *IEEE Transactions on Knowledge and Data Engineering*.

[72] Hamilton, W., Ying, Z., and Leskovec, J. (2017a). Inductive representation learning on large graphs. In *NeurIPS*.

[73] Hamilton, W. L., Ying, R., and Leskovec, J. (2017b). Representation learning on graphs: Methods and applications. *IEEE Data Eng. Bull.*

[74] Harvey, M., Crestani, F., and Carman, M. J. (2013). Building user profiles from topic models for personalised search. In *CIKM*.

[75] Hasanzadeh, A., Hajiramezanali, E., Boluki, S., Zhou, M., Duffield, N., Narayanan, K., and Qian, X. (2020). Bayesian graph neural networks with adaptive connection sampling. In *ICML*.

[76] He, D., Xia, Y., Qin, T., Wang, L., Yu, N., Liu, T.-Y., and Ma, W.-Y. (2016). Dual learning for machine translation. In *NeurIPS*.

[77] He, X., Liao, L., Zhang, H., Nie, L., Hu, X., and Chua, T.-S. (2017). Neural collaborative filtering. In *WWW*.

[78] Heitmann, B. and Hayes, C. (2016). Semstim: Exploiting knowledge graphs for cross-domain recommendation. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, pages 999–1006. IEEE.

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

[80] Hoffmann, R., Zhang, C., Ling, X., Zettlemoyer, L., and Weld, D. S. (2011). Knowledge-based weak supervision for information extraction of overlapping relations. In *ACL-IJCNLP*.

[81] Hoory, S., Linial, N., and Wigderson, A. (2006). Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43(4):439–561.

[82] Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*.

[83] Hsu, W. H., Kennedy, L. S., and Chang, S.-F. (2007). Video search reranking through random walk over document-level context graph. In *MM*.

[84] Hu, G., Zhang, Y., and Yang, Q. (2018). Conet: Collaborative cross networks for cross-domain recommendation. In *CIKM*.

[85] Hu, L., Cao, J., Xu, G., Cao, L., Gu, Z., and Zhu, C. (2013). Personalized recommendation via cross-domain triadic factorization. In *WWW*.

[86] Hu, Z., Dong, Y., Wang, K., and Sun, Y. (2020). Heterogeneous graph transformer. In *WWW*.

[87] Huang, J., Wang, H., Fan, M., Zhuo, A., and Li, Y. (2020a). Personalized prefix embedding for poi auto-completion in the search engine of baidu maps. In *SIGKDD*.

[88] Huang, J.-T., Sharma, A., Sun, S., Xia, L., Zhang, D., Pronin, P., Padmanabhan, J., Ottaviano, G., and Yang, L. (2020b). Embedding-based retrieval in facebook search. In *SIGKDD*.

[89] Huang, P.-S., He, X., Gao, J., Deng, L., Acero, A., and Heck, L. (2013). Learning deep structured semantic models for web search using clickthrough data. In *CIKM*.

[90] Huang, X., Peng, Y., and Yuan, M. (2018). Mhtn: Modal-adversarial hybrid transfer network for cross-modal retrieval. *IEEE transactions on cybernetics*, 50(3):1047–1059.

[91] Ivanov, S. and Burnaev, E. (2018). Anonymous walk embeddings. In *ICML*.

[92] Jeh, G. and Widom, J. (2003). Scaling personalized web search. In *WWW*.

[93] Ji, G., He, S., Xu, L., Liu, K., and Zhao, J. (2015). Knowledge graph embedding via dynamic mapping matrix. In *IJCNLP*.

[94] Jia, C., Wu, B., and Zhang, X. (2020). Dynamic spatiotemporal graph neural network with tensor network. *ArXiv*, abs/2003.08729.

[95] Jia, S., Xiang, Y., Chen, X., Wang, K., and Shijia, E. (2019). Triple trustworthiness measurement for knowledge graph. In *WWW*.

[96] Jiang, S., Hu, Y., Kang, C., Daly Jr, T., Yin, D., Chang, Y., and Zhai, C. (2016). Learning query and document relevance from a web-scale click graph. In *SIGIR*.

[97] Jin, W., Li, Y., Xu, H., Wang, Y., and Tang, J. (2020). Adversarial attacks and defenses on graphs: A review and empirical study. *ArXiv*.

[98] Jolliffe, I. T. (2011). Principal component analysis. In *International Encyclopedia of Statistical Science*.

[99] Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., and Fei-Fei, L. (2014). Large-scale video classification with convolutional neural networks. In *CVPR*.

[100] Kazemi, S. M., Goel, R., Jain, K., Kobyzev, I., Sethi, A., Forsyth, P., Poupart, P., and Borgwardt, K. (2020). Representation learning for dynamic graphs: A survey. *J. Mach. Learn. Res.*, 21:70:1–70:73.

[101] Kim, Y. (2014). Convolutional neural networks for sentence classification. In *EMNLP*.

[102] Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *ICLR*.

[103] Kipf, T. N. and Welling, M. (2017a). Semi-supervised classification with graph convolutional networks. In *ICLR*.

[104] Kipf, T. N. and Welling, M. (2017b). Variational graph auto-encoders. *NeurIPS (workshop)*.

[105] Klicpera, J., Weißenberger, S., and Gunnemann, S. (2019). Diffusion improves graph learning. In *NeurIPS*.

[106] Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37.

[107] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *NeurPIS*.

[108] Lafferty, J., McCallum, A., and Pereira, F. C. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data.

[109] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

[110] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[111] Lee, J. B., Rossi, R. A., Kim, S., Ahmed, N. K., and Koh, E. (2019). Attention models in graphs: A survey. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 13(6):1–25.

[112] Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., and Bizer, C. (2015). Dbpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6:167–195.

[113] Leskovec, J., Kleinberg, J., and Faloutsos, C. (2007). Graph evolution: Densification and shrinking diameters. In *ACM Transactions on Knowledge Discovery from Data*.

[114] Li, P. and Tuzhilin, A. (2020). Ddtcdr: Deep dual transfer cross domain recommendation. In *WSDM*.

[115] Li, Q., Han, Z., and Wu, X. (2018). Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*.

[116] Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. S. (2016). Gated graph sequence neural networks. In *ICLR*.

[117] Li, Z., Liu, H., Zhang, Z., Liu, T., and Xiong, N. N. (2021a). Learning knowledge graph embedding with heterogeneous relation attention networks. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–13.

[118] Li, Z., Wang, X., Li, J., and Zhang, Q. (2021b). Deep attributed network representation learning of complex coupling and interaction. *Knowledge-Based Systems*, 212:106618.

[119] Lian, J., Zhang, F., Xie, X., and Sun, G. (2017). Cccfnet: a content-boosted collaborative filtering neural network for cross domain recommender systems. In *WWW*.

[120] Liao, R., Brockschmidt, M., Tarlow, D., Gaunt, A. L., Urtasun, R., and Zemel, R. S. (2018). Graph partition neural networks for semi-supervised classification. In *ICLR workshop*.

[121] Lin, Y., Liu, Z., Sun, M., Liu, Y., and Zhu, X. (2015). Learning entity and relation embeddings for knowledge graph completion. In *AAAI*.

[122] Lin, Y.-S., Jiang, J.-Y., and Lee, S.-J. (2013). A similarity measure for text classification and clustering. *IEEE transactions on knowledge and data engineering*, 26(7):1575–1590.

[123] Liu, X., Yan, M., Deng, L., Li, G., Ye, X., and Fan, D. (2021). Sampling methods for efficient training of graph convolutional networks: A survey. *IEEE/CAA Journal of Automatica Sinica*, 9(2):205–234.

[124] Liu, Z., Chen, C., Li, L., Zhou, J., Li, X., Song, L., and Qi, Y. (2019). Geniepath: Graph neural networks with adaptive receptive paths. In *AAAI*.

[125] Loni, B., Shi, Y., Larson, M., and Hanjalic, A. (2014). Cross-domain collaborative filtering with factorization machines. In *ECIR*.

[126] Lu, S., Dou, Z., Jun, X., Nie, J.-Y., and Wen, J.-R. (2019). Psgan: A minimax game for personalized search with limited and noisy click data. In *SIGIR*.

[127] Luo, D., Cheng, W., Yu, W., Zong, B., Ni, J., Chen, H., and Zhang, X. (2021). Learning to drop: Robust graph neural network via topological denoising. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*.

[128] Ma, H., Yang, H., King, I., and Lyu, M. R. (2008). Learning latent semantic relations from clickthrough data for query suggestion. In *CIKM*.

[129] Ma, J., Cui, P., Kuang, K., Wang, X., and Zhu, W. (2019). Disentangled graph convolutional networks. In *ICML*.

[130] Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

[131] Man, T., Shen, H., Jin, X., and Cheng, X. (2017). Cross-domain recommendation: An embedding and mapping approach. In *IJCAI*.

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

[133] Meng, C., Mouli, S. C., Ribeiro, B., and Neville, J. (2018). Subgraph pattern neural networks for high-order graph evolution prediction. In *AAAI*.

[134] Meng, Z., Liang, S., Bao, H., and Zhang, X. (2019). Co-embedding attributed networks. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*.

[135] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *ICLR*.

[136] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *NeurIPS*.

[137] Milo, R., Shen-Orr, S., Itzkovitz, S., Kashtan, N., Chklovskii, D., and Alon, U. (2002). Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827.

[138] Monti, F., Boscaini, D., Masci, J., Rodolà, E., Svoboda, J., and Bronstein, M. M. (2017). Geometric deep learning on graphs and manifolds using mixture model cnns. In *CVPR*.

[139] Murphy, R. L., Srinivasan, B., Rao, V., and Ribeiro, B. (2019). Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. In *ICLR*.

[140] Ni, J., Li, J., and McAuley, J. (2019). Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *EMNLP-IJCNLP*.

[141] Nickel, M., Rosasco, L., and Poggio, T. (2016). Holographic embeddings of knowledge graphs. In *AAAI*.

[142] Nickel, M., Tresp, V., and Kriegel, H.-P. (2011). A three-way model for collective learning on multi-relational data. In *ICML*.

[143] Niepert, M., Ahmed, M., and Kutzkov, K. (2016). Learning convolutional neural networks for graphs. In *ICML*.

[144] Ning, W., Cheng, R., Shen, J., Haldar, N. A. H., Kao, B., Huo, N., Lam, W. K., Li, T., and Tang, B. (2021). Reinforced meta-path selection for recommendation on heterogeneous information networks. *arXiv preprint arXiv:2112.12845*.

[145] Ou, M., Cui, P., Pei, J., Zhang, Z., and Zhu, W. (2016). Asymmetric transitivity preserving graph embedding. In *SIGKDD*.

[146] Page, L., Brin, S., Motwani, R., and Winograd, T. (1999). The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab.

[147] Pan, S. J. and Yang, Q. (2009). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359.

[148] Papp, P. A., Martinkus, K., Faber, L., and Wattenhofer, R. (2021). Dropgnn: random dropouts increase the expressiveness of graph neural networks. *Advances in Neural Information Processing Systems*, 34.

[149] Pareja, A., Domeniconi, G., Chen, J. J., Ma, T., Suzumura, T., Kanezashi, H., Kaler, T., and Leisersen, C. E. (2019). Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *AAAI*.

[150] Perez, E., Strub, F., De Vries, H., Dumoulin, V., and Courville, A. (2018). Film: Visual reasoning with a general conditioning layer. In *AAAI*.

[151] Perozzi, B., Al-Rfou, R., and Skiena, S. (2014). Deepwalk: Online learning of social representations. In *SIGKDD*.

[152] Perozzi, B., Kulkarni, V., Chen, H., and Skiena, S. (2017a). Don't walk, skip!: Online learning of multi-scale network embeddings. In *ASONAM*.

[153] Perozzi, B., Kulkarni, V., Chen, H., and Skiena, S. (2017b). Don't walk, skip!: Online learning of multi-scale network embeddings. In *ASONAM*.

[154] Qu, M., Bengio, Y., and Tang, J. (2019). Gmnn: Graph markov neural networks. In *ICML*.

[155] Ricci, F., Rokach, L., and Shapira, B. (2011). Introduction to recommender systems handbook. In *Recommender systems handbook*, pages 1–35. Springer.

[156] Rong, Y., Huang, W., Xu, T., and Huang, J. (2020). Dropedge: Towards deep graph convolutional networks on node classification. In *ICLR*.

[157] Roweis, S. T. and Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326.

[158] Sahebi, S. and Brusilovsky, P. (2015). It takes two to tango: An exploration of domain pairs for cross-domain collaborative filtering. In *RecSys*.

[159] Sahebi, S. and Walker, T. (2014). Content-based cross-domain recommendations using segmented models. In *RecSys*.

[160] Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *WWW*.

[161] Schlichtkrull, M., Kipf, T. N., Bloem, P., Van Den Berg, R., Titov, I., and Welling, M. (2018). Modeling relational data with graph convolutional networks. In *ESWC*.

[162] Schroff, F., Kalenichenko, D., and Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. In *CVPR*.

[163] Schütze, H., Manning, C. D., and Raghavan, P. (2008). *Introduction to information retrieval*, volume 39. Cambridge University Press Cambridge.

[164] Seongjun, Y., Minbyul, J., Raehyun, K., Jaewoo, K., and JKim, H. (2019). Graph transformer networks. In *NeurIPS*.

[165] Sheikh, N., Qin, X., Reinwald, B., Miksovic, C., Gschwind, T., and Scotton, P. (2021). Knowledge graph embedding using graph convolutional networks with relation-aware attention. *arXiv preprint arXiv:2102.07200*.

[166] Shervashidze, N., Schweitzer, P., Van Leeuwen, E. J., Mehlhorn, K., and Borgwardt, K. M. (2011). Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9).

[167] Shi, C., Li, Y., Zhang, J., Sun, Y., and Yu, P. S. (2017). A survey of heterogeneous information network analysis. *IEEE Transactions on Knowledge and Data Engineering*, 29:17–37.

[168] Shuman, D. I., Narang, S. K., Frossard, P., Ortega, A., and Vandergheynst, P. (2013). The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30:83–98.

[169] Sieg, A., Mobasher, B., and Burke, R. (2007). Web search personalization with ontological user profiles. In *CIKM*.

[170] Singh, A. P. and Gordon, G. J. (2008). Relational learning via collective matrix factorization. In *SIGKDD*.

[171] Socher, R., Chen, D., Manning, C. D., and Ng, A. (2013). Reasoning with neural tensor networks for knowledge base completion. In *NeurIPS*.

[172] Sohangir, S. and Wang, D. (2017). Improved sqrt-cosine similarity measurement. *Journal of Big Data*, 4(1):1–13.

[173] Strehl, A., Ghosh, J., and Mooney, R. (2000). Impact of similarity measures on web-page clustering. In *Workshop on artificial intelligence for web search (AAAI 2000)*, volume 58, page 64.

[174] Stretcu, O., Viswanathan, K., Movshovitz-Attias, D., Platanios, E., Ravi, S., and Tomkins, A. (2019). Graph agreement models for semi-supervised learning. In *NeurIPS*.

[175] Suchanek, F. M., Kasneci, G., and Weikum, G. (2007). Yago: a core of semantic knowledge. In *WWW*.

[176] Sun, C., Myers, A., Vondrick, C., Murphy, K. P., and Schmid, C. (2019). Videobert: A joint model for video and language representation learning. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 7463–7472.

[177] Sun, L., Zhang, Z., Zhang, J., Wang, F., Peng, H., Su, S., and Yu, P. S. (2021). Hyperbolic variational graph neural network for modeling dynamic graphs. In *AAAI*.

[178] Sun, Y. and Han, J. (2012). Mining heterogeneous information networks: principles and methodologies. *Synthesis Lectures on Data Mining and Knowledge Discovery*, 3(2):1–159.

[179] Sun, Y., Han, J., Yan, X., Yu, P. S., and Wu, T. (2011). Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *Proceedings of the VLDB Endowment*, 4(11):992–1003.

[180] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *CVPR*.

[181] Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., and Mei, Q. (2015). Line: Large-scale information network embedding. In *WWW*.

[182] Tang, X., Wang, T., Yang, H., and Song, H. (2019). Akupm: Attention-enhanced knowledge-aware user preference model for recommendation. In *SIGKDD*.

[183] Teevan, J., Adar, E., Jones, R., and Potts, M. A. (2007). Information re-retrieval: Repeat queries in yahoo's logs. In *SIGIR*.

[184] Teevan, J., Liebling, D. J., and Ravichandran Geetha, G. (2011). Understanding and predicting personal navigation. In *ICDM*.

[185] Teru, K. K., Denis, E., and Hamilton, W. L. (2020). Inductive relation prediction by subgraph reasoning. In *ICML*.

[186] Thompson, V. U., Panchev, C., and Oakes, M. (2015). Performance evaluation of similarity measures on similar and dissimilar text retrieval. In *2015 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K)*, volume 1, pages 577–584. IEEE.

[187] Trouillon, T., Welbl, J., Riedel, S., Gaussier, e., and Bouchard, G. (2016). Complex embeddings for simple link prediction. In *ICML*, pages 2071–2080. PMLR.

[188] Umadevi, M. (2020). Document comparison based on tf-idf metric. *International Research Journal of Engineering and Technology (IRJET)*, 7(02):1546–1550.

[189] van den Berg, R., Kipf, T. N., and Welling, M. (2017). Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*.

[190] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *NeurIPS*.

[191] Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. (2018). Graph attention networks. In *ICLR*.

[192] Velickovic, P., Fedus, W., Hamilton, W. L., Liò, P., Bengio, Y., and Hjelm, R. D. (2019). Deep graph infomax. In *ICLR*.

[193] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11:3371–3408.

[194] Vu, T., Nguyen, D. Q., Johnson, M., Song, D., and Willis, A. (2017). Search personalization with embeddings. In *ECIR*.

[195] Vu, T., Willis, A., Tran, S. N., and Song, D. (2015). Temporal latent topic user profiles for search personalisation. In *ECIR*.

[196] Wagstaff, E., Fuchs, F. B., Engelcke, M., Posner, I., and Osborne, M. A. (2019). On the limitations of representing functions on sets. In *ICML*.

[197] Waitelonis, J. and Sack, H. (2012). Towards exploratory video search using linked data. *Multimedia Tools and Applications*, 59(2):645–672.

[198] Wang, D., Cui, P., and Zhu, W. (2016). Structural deep network embedding. In *SIGKDD*.

[199] Wang, H., Ren, H., and Leskovec, J. (2021a). Relational message passing for knowledge graph completion. *SIGKDD*.

[200] Wang, H., Zhang, F., Wang, J., Zhao, M., Li, W., Xie, X., and Guo, M. (2018). Ripplenet: Propagating user preferences on the knowledge graph for recommender systems. In *CIKM*.

[201] Wang, H., Zhang, F., Zhang, M., Leskovec, J., Zhao, M., Li, W., and Wang, Z. (2019a). Knowledge-aware graph neural networks with label smoothness regularization for recommender systems. In *KDD*.

[202] Wang, P., Fu, Y., Xiong, H., and Li, X. (2019b). Adversarial substructured representation learning for mobile user profiling. In *SIGKDD*.

[203] Wang, P., Fu, Y., Zhou, Y., Liu, K., Li, X., and Hua, K. (2020). Exploiting mutual information for substructure-aware graph representation learning. In *IJCAI*.

[204] Wang, Q., Mao, Z., Wang, B., and Guo, L. (2017). Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743.

[205] Wang, X., He, X., Cao, Y., Liu, M., and Chua, T. (2019c). KGAT: knowledge graph attention network for recommendation. In *SIGKDD*.

[206] Wang, X., Ji, H., Shi, C., Wang, B., Ye, Y., Cui, P., and Yu, P. S. (2019d). Heterogeneous graph attention network. In *WWW*.

[207] Wang, Y., Liu, Z., Fan, Z., Sun, L., and Yu, P. S. (2021b). Dskreg: Differentiable sampling on knowledge graph for recommendation with relational gnn. *CIKM*.

[208] Wang, Z., Zhang, J., Feng, J., and Chen, Z. (2014). Knowledge graph embedding by translating on hyperplanes. In *AAAI*.

[209] Wei, L., Zhao, H., and He, Z. (2021). Learn layer-wise connections in graph neural networks. *arXiv preprint arXiv:2112.13585*.

[210] White, R. W., Chu, W., Hassan, A., He, X., Song, Y., and Wang, H. (2013). Enhancing personalized search by mining and modeling task behavior. In *WWW*.

[211] Xie, R., Liu, Z., and Sun, M. (2018). Does william shakespeare really write hamlet? knowledge representation learning with confidence. In *AAAI*.

[Xu et al.] Xu, C., Su, F., and Lehmann, J. Time-aware relational graph attention network for temporal knowledge graph embeddings. In *ICLR*.

[213] Xu, C., Su, F., and Lehmann, J. (2021). Time-aware graph neural network for entity alignment between temporal knowledge graphs. In *EMNLP*.

[214] Xu, H., Sang, S., Bai, P., Yang, L., and Lu, H. (2020). Gripnet: Graph information propagation on supergraph for heterogeneous graphs. *arXiv preprint arXiv:2010.15914*.

[215] Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2019). How powerful are graph neural networks? In *ICLR*.

[216] Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.-i., and Jegelka, S. (2018). Representation learning on graphs with jumping knowledge networks. In *ICML*.

[217] Yang, B., Yih, W.-t., He, X., Gao, J., and Deng, L. (2015). Embedding entities and relations for learning and inference in knowledge bases. In *ICLR*.

[218] Yang, C., Xiao, Y., Zhang, Y., Sun, Y., and Han, J. (2020). Heterogeneous network representation learning: Survey, benchmark, evaluation, and beyond. *arXiv preprint arXiv:2004.00216*.

[219] Yang, L., Wu, F., Wang, Y., Gu, J., and Guo, Y. (2019). Masked graph convolutional network. In *IJCAI*.

[220] Yang, Y. and Qi, Y. (2021). Image super-resolution via channel attention and spatial graph convolutional network. *Pattern Recognition*, 112:107798.

[221] Yao, J., Dou, Z., and Wen, J.-R. (2020). Employing personal word embeddings for personalized search. In *SIGIR*.

[222] Ying, R., Bourgeois, D., You, J., Zitnik, M., and Leskovec, J. (2019). Gnn explainer: A tool for post-hoc explanation of graph neural networks. In *NeurIPS*.

[223] Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., and Leskovec, J. (2018). Graph convolutional neural networks for web-scale recommender systems. In *SIGKDD*.

[224] Yoon, M., Gervet, T., Shi, B., Niu, S., He, Q., and Yang, J. (2021). Performance-adaptive sampling strategy towards fast and accurate graph neural networks. In *SIGKDD*, pages 2046–2056.

[225] You, J., Ying, R., and Leskovec, J. (2019). Position-aware graph neural networks. In *ICML*.

[226] Yu, L., Sun, L., Du, B., Liu, C., Lv, W., and Xiong, H. (2021a). Heterogeneous graph representation learning with relation awareness. *arXiv preprint arXiv:2105.11122*.

[227] Yu, T., Yang, Y., Li, Y., Chen, X., Sun, M., and Li, P. (2020). Combo-attention network for baidu video advertising. In *SIGKDD*.

[228] Yu, T., Yang, Y., Li, Y., Liu, L., Sun, M., and Li, P. (2021b). Multi-modal dictionary bert for cross-modal video search in baidu advertising. In *CIKM*.

[229] Yuan, F., Yao, L., and Benatallah, B. (2019). Darec: Deep domain adaptation for cross-domain recommendation via transferring rating patterns. In *IJCAI*.

[230] Yuan, H., Yu, H., Gui, S., and Ji, S. (2020). Explainability in graph neural networks: A taxonomic survey. *arXiv preprint arXiv:2012.15445*.

[231] Yuan, J., Cao, M., Cheng, H., Yu, H., Xie, J., and Wang, C. (2022). A unified structure learning framework for graph attention networks. *Neurocomputing*.

[232] Zaheer, M., Kottur, S., Ravanbakhsh, S., Póczos, B., Salakhutdinov, R., and Smola, A. J. (2017). Deep sets. In *NeurIPS*.

[233] Zhang, C., Song, D., Huang, C., Swami, A., and Chawla, N. V. (2019a). Heterogeneous graph neural network. In *SIGKDD*.

[234] Zhang, L., Ge, Y., and Lu, H. (2020). Hop-hop relation-aware graph neural networks. *arXiv preprint arXiv:2012.11147*.

[235] Zhang, L. and Lu, H. (2020). A feature-importance-aware and robust aggregator for gcn. In *CIKM*.

[236] Zhang, L., Shi, L., Zhao, J., Yang, J., Lyu, T., Yin, D., and Lu, H. (2022a). A gnn-based multi-task learning framework for personalized video search. *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*.

[237] Zhang, L., Song, H., Aletras, N., and Lu, H. (2022b). Node-feature convolution for graph convolutional networks. *Pattern Recognition*.

[238] Zhang, M. and Chen, Y. (2018). Link prediction based on graph neural networks. In *NeurIPS*.

[239] Zhang, M., Cui, Z., Neumann, M., and Chen, Y. (2018a). An end-to-end deep learning architecture for graph classification. In *AAAI*.

[240] Zhang, N., Wang, W., and Wang, L. (2021). Attributed graph alignment. *arXiv preprint arXiv:2102.00665*.

[241] Zhang, S., Yao, L., Sun, A., and Tay, Y. (2019b). Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)*, 52(1):1–38.

[242] Zhang, Y., Qi, P., and Manning, C. D. (2018b). Graph convolution over pruned dependency trees improves relation extraction. *arXiv preprint arXiv:1809.10185*.

[243] Zhang, Y., Xiang, T., Hospedales, T. M., and Lu, H. (2018c). Deep mutual learning. In *CVPR*.

[244] Zhao, J., Wang, X., Shi, C., Hu, B., Song, G., and Ye, Y. (2021). Heterogeneous graph structure learning for graph neural networks. In *35th AAAI Conference on Artificial Intelligence (AAAI)*.

[245] Zhao, J., Zhou, Z., Guan, Z., Zhao, W., Ning, W., Qiu, G., and He, X. (2019a). Intentgc: a scalable graph convolution framework fusing heterogeneous information for recommendation. In *SIGKDD*.

[246] Zhao, W., Wu, X., and Luo, J. (2020). Cross-domain image captioning via cross-modal retrieval and model adaptation. *IEEE Transactions on Image Processing*, 30:1180–1192.

[247] Zhao, Z., Hong, L., Wei, L., Chen, J., Nath, A., Andrews, S., Kumthekar, A., Sathiamoorthy, M., Yi, X., and Chi, E. (2019b). Recommending what video to watch next: a multitask ranking system. In *RecSys*.

[248] Zhen, L., Hu, P., Wang, X., and Peng, D. (2019). Deep supervised cross-modal retrieval. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10394–10403.

[249] Zhou, Y., Dou, Z., and Wen, J.-R. (2020). Encoding history with context-aware representation learning for personalized search. In *SIGIR*.

[250] Zhu, F., Chen, C., Wang, Y., Liu, G., and Zheng, X. (2019). Dtcdr: A framework for dual-target cross-domain recommendation. In *CIKM*.

[251] Zhu, F., Wang, Y., Chen, C., Zhou, J., Li, L., and Liu, G. (2021a). Cross-domain recommendation: challenges, progress, and prospects. In *IJCAI*.

[252] Zhu, Z., Zhang, Z., Xhonneux, L.-P., and Tang, J. (2021b). Neural bellman-ford networks: A general graph neural network framework for link prediction. *ArXiv*, abs/2106.06935.

[253] Zugner, D., Akbarnejad, A., and Gunnemann, S. (2018). Adversarial attacks on neural networks for graph data. In *SIGKDD*.

[254] Zugner, D. and Gunnemann, S. (2019a). Adversarial attacks on graph neural networks via meta learning. In *ICLR*.

[255] Zugner, D. and Gunnemann, S. (2019b). Certifiable robustness and robust training for graph convolutional networks. In *SIGKDD*.