Graph representation learning

Zhang, Xinyi

2022

Zhang, X. (2022). Graph representation learning. Doctoral thesis, Nanyang Technological University, Singapore. <https://hdl.handle.net/10356/155039>

<https://hdl.handle.net/10356/155039>

<https://doi.org/10.32657/10356/155039>

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License (CC BY-NC 4.0).

Downloaded on 26 Apr 2023 21:45:44 SGT

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

GRAPH REPRESENTATION LEARNING

ZHANG XINYI

SCHOOL OF ELECTRICAL & ELECTRONIC ENGINEERING

2022

GRAPH REPRESENTATION LEARNING

ZHANG XINYI

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Doctor of Philosophy

2022

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

21/12/2021

.....

Date

NTU NTU NTU NTU NTU NTU NTU NTU
NTU NTU NTU NTU NTU NTU NTU NTU
Zhang Xinyi
NTU NTU NTU NTU NTU NTU NTU NTU

.....

Zhang Xinyi

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

21/12/2021

.....

Date

NTU NTU NTU NTU NTU NTU NTU NTU
NTU NTU NTU NTU NTU NTU NTU NTU
NTU NTU NTU NTU NTU NTU NTU NTU
NTU NTU NTU NTU NTU NTU NTU NTU

Lihui Chen

.....

Prof. Chen Lihui

Authorship Attribution Statement

This thesis contains material from 2 paper(s) published in the following peer-reviewed journal(s) / from papers accepted at conferences in which I am listed as an author.

Chapter 3 is published as **Xinyi, Zhang**, and Lihui Chen. "Capsule graph neural network." In International conference on learning representations. 2019.

The contributions of the co-authors are as follows:

- A/Prof Chen provided the initial project direction and revised the manuscript.
- I co-designed the study with A/Prof Chen and performed all the experiments and data analysis work at the School of Electrical and Electronic Engineering.

Chapter 4 is published as **X. Zhang** and L. Chen, "mSHINE: A Multiple-meta-paths Simultaneous Learning Framework for Heterogeneous Information Network Embedding," in IEEE Transactions on Knowledge and Data Engineering, doi: 10.1109/TKDE.2020.3025464.

The contributions of the co-authors are as follows:

- A/Prof Chen provided the initial project direction and revised the manuscript.
- I co-designed the study with A/Prof Chen and performed all the experiments and data analysis work at the School of Electrical and Electronic Engineering.

21/12/2021

.....

Date

NTU NTU NTU NTU NTU NTU NTU NTU
NTU NTU NTU NTU NTU NTU NTU NTU
Zhang Xinyi
NTU NTU NTU NTU NTU NTU NTU NTU

.....

Zhang Xinyi

Acknowledgements

Throughout my Ph.D study in Nanyang Technological University (NTU), I have received a great deal of support and assistance.

Firstly, I am extremely grateful to Prof. Chen Lihui and Prof. Jiang Xudong, who encouraged me to start my Ph.D study in a new field for me in the beginning. Especially for Prof. Chen Lihui, as my Ph.D supervisor, she has provided me with valuable and careful advices on both my research and campus life. She gives me with the greatest possible flexibility to explore research topics based on my own interest which makes my challenging Ph.D study become interesting and I'd like to express my deep appreciation to her.

I would like to thank the School of Electrical and Electronics Engineering, NTU, for providing me with the research student scholarship. I would also like to thank the colleagues of Media Technology Laboratory for their support.

My special thanks go to Dr. Annamalai, Dr. Soh Zhan Yi Charlie, and Xu Jiahao for their valuable support and suggestions, and those FYPs from whom I have learnt how to communicate and cooperate.

Then, I wish to thank my families(including Zixu) in Sichuan for their great support over these years. I would also like to thank my friends(including Wang Yixin and Xu Shihao) in Singapore(903) for their company so that I never feel lonely during this journey after their showing up.

Lastly, I would like to thank that young but strong me, as well as the stochastic but lovely world.

Abstract

Graph is a type of structured data which is attracting increasing attention in recent years due to its strong capability in describing multiple objects as well as their relationships. Common graphs include social networks, biological networks, and chemical structures, and so on. Therefore, graph analysis becomes an important research area, and exploring methods to effectively and efficiently process graphs is a valuable research topic. Graph representation learning has been shown to be promising in graph analysis where the graphs or part of the graphs will be converted into a lower-dimensional space while the important information (e.g., structure information and attribute information) of the graph is still well preserved. By feeding these new generated lower-dimensional graph representations to the off-the-shelf machine learning methods(including neural networks), this graph-type of data can be analyzed and then applied in various real-world applications such as link prediction, community detection, node classification as well as graph classification.

Currently, there are two main categories of graph representation learning methods, *graph embedding based methods* and *graph neural network (GNN) based methods*, which have respectively applicable scenarios. Graph embedding based methods aims to embed graph data into lower dimensional space during which the graph representations are learned by preserving predefined proximity. Then, the learned graph representations can be feed to different downstream models for different applications. However, in GNN-based methods, different GNN structures will be separately designed and trained for different applications.

Besides different techniques used in graph representation learning, according to the fineness of the objects for embedding, the graph representation learning methods can be mainly divided into node-level, edge-level, and graph-level representation learning methods. Here, node-level and graph-level representation learning methods have attracted more attention since edge representations can be generated through node representations.

Due to the diversity of graph types, objects of representation learning as well as their corresponding real-world applications, graph representation learning still faces various challenges, so different techniques need to be explored in different specific situations. In graph-level representation learning, the main challenge is how to effectively pool multiple node representations to generate high-quality graph representations where rich information needs to be encoded. However, the low encoding efficiency of scalar-based representations is still a problem. As for the node-level graph representation learning, traditional node-level representation learning methods focus more on homogeneous information networks (homogeneous graphs) and have achieved remarkable performance. However, when it takes different aspects of a graph into consideration, especially in heterogeneous information networks (HINs) where multiple types of nodes and edges should be considered, node-level representation learning methods designed for homogeneous information networks are suboptimal and fail to handle the semantic incompatibility brought from HIN.

To address the problems mentioned above, this thesis introduces three research works to improve the current graph representation learning methods:

1. We first focus on GNN-based methods and propose Capsule Graph Neural Network (CapsGNN), which is a novel message-passing-based architecture for graph-level representation learning. In this work, the scalar node representations are firstly generated through normal GNNs and then used to generate graph-level representations. Generating a high-quality graph representation should preserve much information such as node position, subgraph direction and node connection, however, encoding this information in the form of scalar means activating elements in a vector one-by-one which is exponentially less efficient than encoding them with distributed representations, so routing mechanism is applied to generate graph-level representations in the form of capsules which can encode graph information more efficiently. An attention module and a coordinate addition module are also introduced to improve the quality of the generated graph representations which is evaluated through the graph classification result. Experiments are also conducted to verify the efficiency of the use of capsules in graph representation learning.
2. Then we look into the graph embedding based methods and focus on node-level representation learning methods especially for the HIN. HIN is a special type of information network (graph) with a stronger description capability and it contains various types of nodes and edges. However, the heterogeneity of HINs also raises

the problem of potential semantic incompatibility which might affect the performance of traditional node representation learning algorithms in downstream tasks. To address this issue, a novel graph embedding based HIN representation learning framework named mSHINE is designed to simultaneously learn multiple node representations for different meta-paths (aspects) of the HIN. At the same time, the proposed shared node representations in mSHINE allow compatible information to be shared across different meta-paths which improves the learning efficiency and the learned node representation quality. A set of criteria is also proposed for selecting initial meta-paths for mSHINE which is helpful to reduce the meta-path selection cost. This work is evaluated through node classification and link prediction tasks.

3. By analyzing the advantages and disadvantages of the previously proposed work which include both GNN-based and graph embedding based methods, a novel HIN embedding method named eHIN is proposed. eHIN is designed to address the existing problems in current node representation learning algorithms (including mSHINE) for HIN by combining the techniques from the GNN-based methods. Besides the shared node representations which is proposed from mSHINE, eHIN also applies the convolution-based module and the non-sampling optimization technique to improve the learning efficiency further. The convolution-based module is designed to capture the spatial information of nodes which is realized through the attention mechanism and the meta-path-based sequential information of nodes is preserved through a GRU unit. Node attributes and label information will be preserved in eHIN which shows an important role in some real-world applications. Systematically analyses are conducted to show the learning efficiency of eHIN and the properties of node representations learned from different meta-paths.

In each work, we also discussed the performance of the learned graph representations in the corresponding applications such as graph classification, link prediction as well as the node classification, and the proposed methods show promising performance on almost all the tested applications and datasets.

Contents

Acknowledgements	i
Abstract	ii
List of Figures	viii
List of Tables	xi
Symbols and Acronyms	xii
1 Introduction	1
1.1 Motivation and Goals	3
1.2 Major Contributions	5
1.3 Thesis Outline	9
2 Literature Review	11
2.1 Non-learnable Methods	11
2.2 Non-deep Learning Methods	13
2.2.1 Graph Embedding based Methods	13
2.2.2 Others	14
2.3 Deep Learning Method	15
2.3.1 Graph Neural Network based Methods	15
2.3.2 Others	17
3 Preliminaries	19
3.1 Notation and Definition	19
3.2 Common Datasets	21
4 Capsule Graph Neural Network	24
4.1 Introduction	24
4.2 Preliminary and Related Work	26
4.2.1 Problem statement	26
4.2.2 Graph Neural Network	26
4.2.3 Capsule Neural Network	27

4.2.4	Attention Mechanism	28
4.3	Methodology	28
4.3.1	Overall Architecture	28
4.3.2	Basic Node Capsules	29
4.3.3	High-Level Graph Capsules	30
4.3.4	Classification	32
4.4	Differences from Existing Approaches	34
4.5	Experiments	35
4.5.1	Dataset	36
4.5.2	Baselines Methods	36
4.5.3	Experimental Set-up	37
4.5.4	Classification Result	38
4.5.5	Efficiency of Capsules	39
4.5.6	Graph Capsules Analysis	41
4.5.7	Attention Analysis	43
4.5.8	Contribution of Each Module	44
4.6	Conclusion	46
5	mSHINE: A Multiple-meta-paths Simultaneous Learning Framework for Heterogeneous Information Network Embedding	47
5.1	Introduction	47
5.2	Preliminary	49
5.2.1	Related Concepts	50
5.2.2	Problem Statement	50
5.3	Related Work	51
5.3.1	Negative Sampling	51
5.3.2	Recurrent Neural Network	52
5.4	Methodology	52
5.4.1	Node Representation Learning	53
5.4.1.1	Structure Information Modelling	54
5.4.1.2	Semantic Information Modelling	55
5.4.1.3	Optimization Objective	57
5.4.2	Initial Meta-path Selection	59
5.5	Experiments	61
5.5.1	Datasets	61
5.5.2	Baseline Methods	62
5.5.3	Implementation Details	63
5.5.4	Node Classification	64
5.5.4.1	Experimental setup	64
5.5.4.2	Experimental results and discussion	66
5.5.5	Link Prediction	69
5.5.5.1	Experimental setup	69
5.5.5.2	Experimental result and discussion	71

5.6	Conclusion	73
6	eHIN: An Efficient Meta-path Sharing Heterogeneous Information Network Representation Learning Method	75
6.1	Introduction	75
6.2	Preliminaries	77
6.3	Methodology	79
6.3.1	Overall Framework	81
6.3.2	Node Representation Generation	82
6.3.2.1	Node Attribute Representation (Optional)	82
6.3.2.2	Static Meta-path-based Node Representation	82
6.3.2.3	Temporal Meta-path-based Node Representation	83
6.3.3	Objective Optimization	85
6.3.3.1	Static Node Representation Updating	85
6.3.3.2	Heterogeneous Connection Information Preservation	85
6.3.3.3	Label Information Preservation	88
6.3.3.4	Overall Objective Function	89
6.3.4	Representation Generation for Downstream Tasks	89
6.4	Experiments	89
6.4.1	Datasets	89
6.4.2	Methods to Compare	91
6.4.3	Implementation Details	91
6.4.4	Node Classification	92
6.4.4.1	Experimental setup	92
6.4.4.2	Experimental results and discussion	94
6.4.5	Link Prediction	95
6.4.5.1	Experimental setup	95
6.4.5.2	Experimental results and discussion	97
6.4.6	Ablation Study	98
6.4.7	Meta-path Information Analysis	99
6.4.8	Comparison with mSHINE	103
6.5	Conclusion	104
7	Conclusion and Future Research	106
7.1	Summary of completed work	106
7.2	Future Work	107
	Author's Publications	109
	Bibliography	110

List of Figures

1.1	Examples of Real-World Data Represented by Graphs.	2
1.2	Research roadmap.	5
2.1	Variants of message-passing-based GNN model.	16
3.1	A example of normal graph.	19
3.2	A simple example of HIN (IMDB information network) and the corresponding network schema.	20
4.1	Framework of CapsGNN.	29
4.2	The structure of Attention Module in CapsGNN.	31
4.3	The structure of Coordinate Addition Module.	32
4.4	The structure of Reconstruction Loss Module.	33
4.5	Comparison of efficiency in features representation. The horizontal axis represents d_n-d_g-P . The vertical axis represents classification accuracy on NCI1.	40
4.6	The attention distribution on IMDB-M dataset and REDDIT dataset.	44
5.1	The structure of a simple RNN unit.	52

5.2	The schematic illustration of mSHINE. Meta-path selection criteria are applied to select valid meta-paths first and these valid meta-paths are used to direct sampling three-element node sequences. For each three-element node sequence $n_{i-1} \xrightarrow{T_a^e} n_i \xrightarrow{T_b^e} n_{i+1}$, the general basic node representation $\vec{x}_{n_i}$ for n_i and general state node representations $\vec{h}_{n_{i-1}}$, $\vec{h}_{n_i}$ for n_{i-1} and n_i are generated through looking up the basic/state node representation tables. Then meta-path-related functions are applied to process the general basic/state node representations to generate $\vec{x}_{n_i}^{M_j}$, $\vec{h}_{n_{i-1}}^{M_j}$ and $\vec{h}_{n_i}^{M_j}$ respectively. The processed node representations ($\vec{x}_{n_i}^{M_j}$ and $\vec{h}_{n_{i-1}}^{M_j}$) are then fed into RNN-inspired unit together with the edge representation $\vec{e}_{T_{ab}^e}$ to generate a new state node representation $\vec{h}_{n_i, T_{ab}^e}^{M_j}$ for n_i as well as the prediction of the next node n_{i+1} . All the trainable parameters in the proposed structure are trained based on $\mathcal{L}_{state}(n_i)$ and $\mathcal{O}(n_{i+1})$. Finally, the basic node representations as well as meta-path-related functions can be used to generate meta-path-related node representations and the trained node representation learning module can be applied in downstream link prediction tasks.	53
5.3	An example of decomposing a meta-path into the corresponding triple node sequence type set.	59
6.1	An example of HIN and network schema. a)An example of CORA information network and a_i, p_i, t_i denote Authors, Papers and terms respectively. b)The network schema of CORA. Three types of nodes and edges are displayed where A, P, T denote the node type of Author, Paper and Term respectively.	78

6.2	Overview of eHIN where different node types are denoted with different colors. a) At each iteration, based on the meta-path information(i.e., the current meta-path (M_1), the corresponding triple sequence type ($Rel_2:A \rightarrow P \rightarrow T$) and the negative meta-path (M_3)), the training samples can be prepared accordingly: the center node (p_1), the corresponding head neighbors ($\mathcal{N}^A(p_1) = \{a_1, a_2\}$) and tail neighbors ($\mathcal{N}^T(p_1) = \{t_1, t_3\}$). b) Before generating the temporal node representation of p_1 , the attribute node representation $\vec{x}_{p_1}$ of p_1 and the static node representations $\vec{h}_{p_1}^s, \{\vec{h}_{a_1}^s, \vec{h}_{a_2}^s\}$ of $p_1, \mathcal{N}^A(p_1) = \{a_1, a_2\}$ should be prepared through transforming as well as looking up the static node representation matrix(H^s) respectively. In this figure, we use two red vectors to represent the generated $\{\vec{h}_{a_1}^s, \vec{h}_{a_2}^s\}$. Then the temporal node representation $\tilde{h}_{p_1, A \rightarrow P}^{(1)}$ is generated based on $\vec{x}_{p_1}$ together with $\vec{h}_{p_1}^s$ and $\{\vec{h}_{a_1}^s, \vec{h}_{a_2}^s\}$ (Eq.6.3). $\tilde{h}_{p_1, A \rightarrow P}^{(1)}$ is then considered as the ground-truth for updating the static node representation of p_1 ($\mathcal{L}_1^{M_1}$:Eq.6.12); c) In the stage of optimization, the temporal and static node representations of p_1 are used to predict the tail neighbors ($\mathcal{L}_2^{M_1}$: Eq.6.16) and labels ($\mathcal{L}_4^{M_1}$:Eq.6.23). To achieve this, the static node representations($\{\vec{h}_{t_1}^s, \vec{h}_{t_2}^s\}$) of $\{t_1, t_2\}$ should be prepared respectively in advance which are noted with two green vectors here in the figure. Concurrently, the prediction value of the connection between p_1 and its tail neighbors($\{t_1, t_3\}$) which is calculated from the negative meta-path (M_3) should be minimized and the corresponding loss is $\mathcal{L}_3^{M_1}$ (Eq.6.22).	80
6.3	The generation of $\tilde{h}_{nc}^{(j)}$ and $\vec{h}_{nc}^{(j)}$ and their relationship	84
6.4	Classification performance(<i>f1-weighted</i>) comparison on Douban Movie 2 between simultaneously learned and separately learned meta-path-based node representations.	99
6.5	Classification performance(<i>f1-weighted</i>) comparison on Cora between simultaneously learned and separately learned meta-path-based node representations.	100
6.6	(a),(c),(e) The classification performance(<i>f1-weighted</i>) of node representations learned from single meta-path and meta-path combination on Douban Movie, CORA, IMDB-S. (b),(d),(f) The changes of <i>f1-weighted</i> in classification performance caused by simultaneously learning node representations with other meta-paths on Douban Movie, CORA, IMDB-S.	102
6.7	Classification performance comparison between eHIN and mSHINE along epochs. (a),(b) <i>f1-macro</i> and <i>f1-micro</i> on Douban Movie 2. (c),(d) <i>f1-macro</i> and <i>f1-micro</i> on CORA.	103

List of Tables

3.1	Description of Datasets for Graph-level Representation Learning . .	21
3.2	Statistics of The Experimental Datasets	22
4.1	Experiment Result of Biological Dataset	38
4.2	Experiment Result of Social Dataset	38
4.3	Details of Tested Architectures in Efficiency Evaluation Experiment	40
4.4	Visualization of Graph Capsules	42
4.5	Visualization of Class Capsules	42
4.6	Comparison of Each Module	45
5.1	The Number of Initial Meta-paths for Each Dataset	64
5.2	Performance of Node Classification	66
5.3	Performance of Node Classification with Yelp with various Meta-paths(Aspects)	68
5.4	Performance of Node Classification with Cora with Various Window Size from $w_s = 1$ to $w_s = 3$	69
5.5	Performance of Link Prediction	71
5.6	Performance of Link Prediction Based on Different Meta-path (Aspects)	73
6.1	An Example of the Meta-path Selection Result	78
6.2	Statistics of Extracted Meta-paths for Each Dataset	92
6.3	Performance of Node Classification	93
6.4	Performance of Link Prediction	96
6.5	Classification Performance for Ablation Study in eHIN	99

Symbols and Acronyms

Symbols

$\mathcal{G}$	graph
n_i/e_i	the i-th node/edge
$\mathbb{V}/\mathbb{E}$	node/edge set
T^v/T^e	node/edge type
$\mathbb{T}^v/\mathbb{T}^e$	node/edge type set
$\mathcal{N}(n_i)$	neighbor node set of n_i
$\mathcal{N}^{T^v}(n_i)$	neighbor node set of n_i with the node type of T^v
$\mathcal{T}_{\mathcal{G}}$	network schema of graph $\mathcal{G}$
M_i	the i-th meta-path
$\mathbb{M}$	meta-path set
$\vec{e}_{(T_p^v \rightarrow T_c^v)}$	edge type embedding
$T_{E_{ab}}/Rel_r$	triple sequence type
$\vec{r}_{T_{E_{ab}}}$	relation representation of three-element node sequence with the type of $T_{E_{ab}}$
$Rel^{M_i}/Rel^{\mathcal{G}}$	decomposed triple sequence type set of $M_i/\mathcal{G}$
$\mathbb{R}^n$	the n -dimensional Euclidean space
$\circ$	the Hadamard (component-wise) product

Acronyms

GNN	Graph Neural Network
CapsNet	Capsule Neural Network
HIN	Heterogeneous Information Network

Chapter 1

Introduction

Graph, also referred to as information network, is a type of data that can be used to explicitly describe a set of objects and their relationships. Graph is widely used in recent years in various fields and the common graphs include physic systems [1], social relationships, chemical structures [2], knowledge graphs [3], and so on. Some examples of the real-world graphs are shown in Figure 1.1. The wide use of graph makes graph analysis (its practical applications include recommendation systems [4], community detection [5], malware detection [6], transportation system [7] etc.) become a ubiquitous and necessary topic in both the research community and industry sectors. Considering that machine learning, including deep learning, techniques have reached relatively good performance in many areas, representing graphs or parts of graphs with lower-dimensional vectors and then analyzing their properties with the off-the-shelf machine learning (including deep learning) techniques for graph analysis is a feasible direction. However, this requires that the lower-dimensional vectors used to represent graphs or parts of graphs be able to well preserve the important information of the original graphs.

In the beginning, some approaches have been proposed to encode graphs through user-defined features such as degree statistics, kernel functions [8–10]. However, regardless of the extensive human effort required by manually exploring useful graph features, it is hard for us to find all the important graph features for different datasets with various properties, still less there are varied applications that may have different requirements of the extracted graph features. So, a number of

FIGURE 1.1: Examples of Real-World Data Represented by Graphs.

techniques are proposed to allow to automatically find the representative and lower-dimensional graph features through learning-based algorithms. These techniques are referred to as graph representation learning methods and the learned lower-dimensional vectors are referred to as graph representations or graph embeddings. Currently, there are two main categories of methods for graph representation learning [11, 12]: *graph embedding methods* [13–15] and *graph neural network (GNN)* based methods¹ [16–18].

Other than the techniques used in graph representation learning, according to the fineness of the objects, current graph representation learning can also be divided into different levels: node-level, edge-level, and graph-level representation learning methods. People focus more on the node-level and graph-level representation

¹There are other categories of methods such as generative methods, kernel-based methods, etc. while this thesis only focus on these most attractive two categories.

learning methods since the edge representations can actually be generated through node representations in most cases.

In the field of graph analysis, due to the diversity of graph types, objects for representation learning as well as their corresponding real-world applications, whether the graph embedding methods or the GNN-based methods, graph representation learning still faces various challenges, so different techniques need to be explored in different specific situations.

1.1 Motivation and Goals

Based on our observations of the current works in graph representation learning, we identify several problems that need to be addressed in graph embedding methods or GNN-based methods and they are listed as below:

- **The low encoding efficiency of scalar features.** This problem mainly exists in graph-level graph representation learning and GNN-based methods. The high-quality node embeddings learned from the GNNs have been applied to a wide range of node-based applications and some of them have achieved state-of-the-art performance. However, when applying node embeddings learned from GNNs to generate graph embeddings, the scalar node representation generated from GNNs may not suffice to preserve the node/-graph properties efficiently, resulting in sub-optimal graph embeddings [19]. Specifically, to generate better representations for the whole graphs, it is required to design an effective pooling module to preserve the rich property information (including direction, position information, etc.) extracted from individual nodes or edges. However, encoding these rich property information in the form of scalar, which is the most common way, means activating elements in a vector one-by-one which is exponentially less efficient than encoding them with distributed representations. So it is critical to explore an effective pooling module to generate the graph representations with rich information and high encoding efficiency.
- **Limitation of receptive field.** The receptive field of a GNN is usually limited by its number of layers and this is inefficient in memory consumption

[20] and harmful to the learned representation quality especially when cooperating with meta-path to process the heterogeneous information networks (HINs) (HIN is also one of the focus graph types in this thesis). Meta-path [21–24] is one of the most common techniques used for dealing with the heterogeneity of HIN where each meta-path defined in a HIN is considered as a separate aspect. When modeling a long meta-path with a GNN, the GNN is required to have a large receptive field to correspond to the long meta-path, which then leads to a deep GNN. Then the memory consumption of each round of parameter update in GNN may become large because of the large number of GNN layers. Even when mini-batch update [25] is applied, the neighborhood size still grows exponentially with the number of GNN layers and this is inefficient in terms of training space. In practical use, people choose to limit the number of GNN layers which results in the limitation of the length of meta-paths that can be processed. This is not optimal in terms of the quality of the learned representations.

- **Aspects selection.** The information network is widely used to describe complex relationships among different objects and there is a common assumption that there are different aspects existing in an information network where some of them are compatible with each other while others are not. So, learning the node representations while ignoring the incompatibility of those aspects is actually harmful to the learned node quality especially when processing HINs where there are multiple types of nodes or edges [26, 27]. To handle the incompatible aspects that existed in HINs, before directly learning node representations, the first step should be to detect different aspects from the HIN in advances. However, no matter from the perspective of the GNN-based algorithms or the graph embedding methods, there are a limited number of works done for automatically finding aspects for representation learning from a given HIN. Besides detecting aspects, being able to analyze the relationship among the found aspects is also important, which helps to explore more suitable node representation learning modules to handle the found aspects and get interpretable (to some extend) results.
- **Shared information among different aspects in HINs.** When processing HIN, because of the incompatibility among some aspects, the node representations might need to be separately learned for each node within

different aspects. However, the detected aspects might not be totally independent to each other, so learning node representations while totally ignoring the correlation among those aspects is not efficient and is also harmful to the quality of the learned node representations neither. This problem is currently widely existing in many graph embedding methods, therefore, designing HIN representation learning methods that are able to well make use of the shared information among different aspects is necessary.

- **Topological information preserving.** Topological information around each node is the most basic and critical information for describing the corresponding node. However, most graph embedding methods proposed to preserve this topological information by converting them into sequential information, which requires: 1) a large number of node sequences to be generated around each node [22, 28]; or 2) recurrently to apply the sequential information processing unit (e.g., RNN) to each node. Compared with GNN-based methods, these methods are inefficient in terms of capturing topological information.

1.2 Major Contributions

FIGURE 1.2: Research roadmap.

Three main works have been completed to address the problems mentioned above and their relationships are summarized in Figure 1.2. First, we propose a novel GNN-based structure, CapsGNN, for learning representations for the whole graph,

where the concept of capsule [19] is introduced to improve the encoding efficiency. Then, we propose a novel HIN representation learning structure (mSHINE) based on graph embedding based method to handle the heterogeneity of HINs. A meta-path selection module is also proposed in mSHINE to address the problem of aspect selection. Besides, the compatible information among different aspects can be fully shared in mSHINE. At last, by combining the techniques used in CapsGNN(attention pooling) and mSHINE(meta-path selection), we propose a more advanced HIN representation learning model, where the topological information can be more efficiently captured compared with mSHINE, and at the same time, the long meta-path sequential information can be well modeled as well. So basically, our study could cover different levels(both node and graph-level representation learning) and different basic directions(graph embedding and GNN-based methods) of graph representation learning.

More details of these three works are shown below:

1. **CapsGNN.** To address the problem of low encoding efficiency in learning representations for the whole graphs based on node representations generated from GNN, we designed a novel GNN-based structure where the concept of capsule [29] is introduced to extend the scalar-based features to distributed representations, and the dynamic routing mechanism [19] is applied to generate the graph representations effectively.

Corresponding Contribution:

- We design a novel architecture, capsule graph neural network (CapsGNN), for direct graph processing where the concept of capsule is introduced to improve the encoding efficiency. Through CapsGNN, multiple graph representations will be learned for each graph. To the best of our knowledge, this work is the first to apply the capsule technique to graph representation learning and generate graph features in the form of distributed representations.
- The performance of extracted graph embeddings from CapsGNN has validated on classification tasks on 10 real-world datasets (including 5 biological datasets and 5 social datasets), and it achieves state-of-the-art performance on 6 out of 10 benchmark datasets and comparable performance on the rest.

- Comprehensive experiments are conducted to show the properties of each module used in the proposed architecture. Besides, T-SNE [30] is used to visualize the learned graph embeddings and the results show that different graph capsules indeed capture different information of the graphs.
2. **mSHINE**. To handle the heterogeneity of HINs, we propose a novel HIN representation learning framework named mSHINE (**m**ultiple meta-paths **S**imultaneous learning framework for **H**eterogeneous **I**nformation **N**etwork **E**MBEDDING) which is developed based on graph embedding based methods. mSHINE contains two important components: meta-path selection module and node representation learning module. Specifically, given a HIN, a set of criteria proposed in mSHINE is firstly applied to select valid meta-paths for training. Then the node representation learning module, which is inspired by the recurrent neural network (RNN) [31], is applied to iteratively capture node environment information within each meta-path and this module is trained by preserving connectivity proximity. During this procedure, the proposed shared node representations allow the information among different meta-paths to be flexibly shared. At last, the node representations learned for different meta-paths can be easily generated by looking up the well trained node representation tables. Then users can choose the most suitable meta-path based on the specific tasks.

Corresponding Contribution:

- We proposed a novel HIN representation learning framework that includes two components: the initial meta-path selection module and the node representation learning module. The effectiveness of the proposed mSHINE framework is verified via extensive experiments (including node classification and link prediction) study on 5 real-world datasets.
- An initial meta-path selection module, which consists of a set of meta-path selection criteria, is proposed to select valid meta-paths for training. This module is helpful to reduce the cost of meta-path (aspect) exploration.

- The proposed node representation learning module is designed to allow to simultaneously learn node representations for multiple selected meta-paths. Besides, the shared node representations facilitate the sharing of information across different meta-paths, which is helpful to improve the learning efficiency and the quality of the learned node representations.
 - The information introduced by meta-path can be naturally integrated through the RNN-inspired structure which thereby handles the incompatibility brought by the heterogeneity of HINs to some extent.
3. **eHIN**. We propose another HIN representation learning method named eHIN (an **efficient** meta-path sharing **H**eterogeneous **I**nformation **N**etwork representation learning method) where the related techniques used in CapsGNN and mSHINE are introduced and combined to further improve the learning efficiency and the learned node representation quality. Given a HIN as well as a set of meta-paths which are selected by the criteria proposed in mSHINE, eHIN makes use of the convolution-based unit cooperating with the attention mechanism, which is widely used in GNN-based methods, to simultaneously capture the topological information around nodes as well as the sequential information introduced by meta-paths, based on which we can generate the temporal node representations, which are then considered as the ground truth for training the corresponding static meta-path-based node representations. Multiple carefully designed objective functions are applied to training node representations by preserving the connectivity proximity and the label information. At last, the well-trained static meta-path-based node representations in eHIN can be easily generated by combining the well-trained shared node representation and the corresponding view generation function, which simplifies its use in the downstream applications.

Corresponding Contribution:

- Proposed eHIN, a novel HIN representation learning method with carefully designed optimization functions to integrate the connectivity information and the label information, whose effectiveness is verified via comprehensive experiments study (i.e., node classification and link prediction) on 5 real-world datasets.

- To improve the node representation quality, we proposed to preserve both spatial and meta-path-based sequential information when generating the temporal node representations. This captured spatial and sequential information can be then integrated into the corresponding static node representations by iteratively updating these static node representations with the generated temporal node representations. After training, the static node representations can be more easily generated through the well trained model and this can simplify its use in downstream applications.
- The shared node representations used in this work can facilitate the compatible information across different meta-paths to be shared, which improves the learning efficiency and representation quality. We also conducted experiments to verify this.
- The applied convolution-based unit and meta-path-based non-sampling optimization methods are capable of capturing the topological information more efficiently.

1.3 Thesis Outline

The remaining of the thesis is organized as follows:

In Chapter 2, we present the related work on graph representation learning from three perspectives: non-learnable methods, non-deep learning methods, and deep learning methods where the characteristics and limitations of different types of methods are introduced in detail.

In Chapter 3, we provide the basic notations used for describing different types of graphs for easy reading. Besides, we also introduced some widely used experimental datasets of different graph categories

Chapter 4 presents our work for graph-level representation learning. Specifically, a novel graph neural network architecture named CapsGNN is proposed to learn the representations for the whole graphs. The concept of capsules introduced and the dynamic routing mechanism is applied to generate multiple graph representations for each graph in the form of distributed representations. The performance of the

learned graph representation has validated on classification tasks on 10 real-world datasets and it achieves state-of-the-art performance on 6 out of 10 benchmark datasets and comparable performance on the rest. The properties of different components in CapsGNN are also thoroughly analyzed through comprehensive experiments.

In Chapter 5, we present mSHINE, a novel HIN representation learning framework, which mainly contains two key components: meta-path selection module and node representation learning module. The effectiveness of mSHINE is also verified in this chapter via extensive experiments (including node classification and link prediction) study on 5 real-world datasets.

In Chapter 6, we present another HIN representation learning method called eHIN which is developed based on mSHINE. The techniques which are widely used in GNN-based methods are incorporated here to improve the learning efficiency. The effectiveness of eHIN is evaluated through node classification and link prediction tasks. Moreover, comprehensive experiments are conducted to analyze the relationships among node representations learned from different meta-paths as well as the improvement of learning efficiency.

In Chapter 7, the works presented in this thesis are concluded and the potential directions in the future are discussed.

Chapter 2

Literature Review

In this section, we give a general introduction of current works done for graph representation learning. In particular, **non-learnable methods** including graph kernel-based methods and matrix factorization based methods, which were the dominant techniques for graph analysis and laid the foundation for many subsequent effective methods, are firstly introduced in Section 2.1. Then we introduced the **non-deep learning methods** (Section 2.2) and **deep learning methods** (Section 2.3) whose main categories are graph embedding based methods (Section 2.2.1) and GNN-based methods (Section 2.3.1) respectively.

2.1 Non-learnable Methods

There are two main non-learnable methods for getting the graph representations: kernel-based methods and matrix factorization and they are introduced as below:

1. **Kernel-based Algorithms.** Kernel-based algorithms proposed to compare each pair of graphs by comparing their sub-components (convolution kernel[8]). Typically, kernel-based algorithms first decompose graphs into sub-components based on the kernel definition, then build graph representations in a feature-based manner. Lastly, some kernel-based machine learning algorithms (i.e., kernel regression, kernel PCA, SVM [32]) are applied to perform graph comparison. The typical kernel-based algorithms for embedding the whole graphs including Weisfeiler-Lehman subtree kernel (WL) [9], the

graphlet count kernel(GK) [33], and the Random Walk kernel(RW)[10] which respectively extract subtree patterns, graphlets and random walks of graphs to represent graphs. As introduced above, kernel-based methods are mainly applied for processing the whole graphs.

2. **Matrix Factorization.** Matrix factorization¹ mainly serves as a technique for dimensionality reduction during the procedure of generating graph representations. Based on the information that wants to preserve, different types of matrices can be selected for factorization. The purpose of matrix factorization for generating graph representations is assigning each node one or more proper representations, and these representations are capable to reconstruct the defined target matrix. So matrix factorization are mainly designed for generating node-level representations. A well-known algorithm is Laplacian eigenmaps(LE) [34] which proposes to calculate the eigenvectors of the graph Laplacian matrix and this algorithm equivalent to using the euclidean distance between node embeddings to model their relationship in the graph. Graph Factorization(GF) [35] directly chooses to reconstruct the adjacency matrix with the dot product of node representations. To preserve higher-order similarity between nodes, GraRep [36] proposes to reconstruct various powers of the adjacency matrix.

Non-learnable methods introduced above either require carefully manually designed graph patterns [9, 10, 33] or are computationally expensive [34, 36] especially when processing huge graphs. Besides, these methods are hard to handle graphs with attributes information or heterogeneous information. So, more flexible and powerful methods for automatically generating graph representations need to be explored and we introduce some of them below.

¹As many learnable proximity-preserving methods, which will be introduced in the next section, can also be explained as matrix factorization, the matrix factorization we discuss in this section only includes non-learnable matrix-factorization-based methods.

2.2 Non-deep Learning Methods

2.2.1 Graph Embedding based Methods

Graph embedding based methods have shown their effectiveness in graph analysis, where lower-dimensional nodes representations are firstly automatically learned by preserving different types of proximities and then fed to the off-the-shelf machine learning models for the downstream applications. More detailed introduction of graph embedding methods are given based on different categories:

Random Walk Methods. The proximity that random-walk-based methods propose to preserve is collected through the random walk and it is normally applied for node-level representation learning. For example, In DeepWalk[13], the most representative method of this category, a number of paths are firstly sampled with random walk. Then Word2vec[37], an effective distributed representation learning algorithm, is applied to embed each node into a lower-dimensional space. During the procedure of learning, each path is considered as a sentence and each node is considered as a word of this sentence. Besides, hierarchical-softmax[38] or negative-sampling[37] is usually applied during the procedure of training to improve the training efficiency. Node2vec[14] follows the same idea except using a more flexible exploration strategy when sampling node sequences. Many other similar variants[4, 39–43] are also proposed according to the specific application scenarios.

Random-walk-based methods[22, 28, 44] are also wildly applied to process HINs due to their easy collaboration with meta-paths (one of the most common techniques for handling the heterogeneity of HINs). For example, Metapath2vec[22], which is developed based on DeepWalk, proposed to conduct the random walk under the direction of meta-path definition. However, these random-walk methods proposed to capture the topological information, known as spatial information, around each node by converting it into sequential information which requires a large number of node sequences to be sampled. This is not efficient when compared with the convolution-based unit which is commonly used in GNN-based methods and we will introduce it in the following section.

First/Second-order Proximity Preserving Methods. Another category of node-level representation learning methods is proposed to explicitly preserve the

first or second-order proximity of the original graphs. (e.g, LINE[15], PTE[45], HNE[46], graph-likelihood[47], and HEBE[48], AspEM[49]). For example, compared with DeepWalk where no clear objective about what network properties are preserved, LINE is designed to preserve both of the first-order and second-order proximities in graphs and it doesn't rely on random walk. When it comes to HIN, usually aspects, which are represented with a single edge type or edge type combination, are detected first, and then methods such as LINE are applied to learn node representations for each aspect[45, 49]. For example, PTE[45] is designed for heterogeneous text networks which makes use of LINE as the representation learning algorithm while it only preserves the second-order proximities. Comparing with random-walk approaches, this category of methods do not require to sample a large number of node sequences which is more efficient for training, however, they show worse performance in capturing the "longer" relationship between nodes[28].

2.2.2 Others

There are still a number of methods that are designed by combining the non-deep learning techniques with the non-learnable methods. In [50], it has been proved that skip-gram, a type of training strategy introduced from Word2vec, with negative-sampling is actually implicitly factorizing a word-context matrix and the cells of this matrix are the pointwise mutual information (PMI) of the word-context pairs which is shifted by a global constant. Inspired by this work, TADW[51] proves that DeepWalk is also equivalent to matrix factorization and text features of nodes can be embedded into the learned node representation under the framework of matrix factorization. Besides, inspired by the kernel-based algorithms, some methods[52, 53] are proposed to apply Word2vec to learn the representations of sub-components of graphs which then can be used to calculate the similarity between each pair of sub-components. Lastly, the learned graph kernels can be applied in some kernel-based machine learning algorithms to do graph classification. By combining Doc2vec[54] with kernel-based methods, some methods[55–57] are proposed to directly learn the representations of the whole graphs. So far, all the algorithms introduced above, including non-learnable methods as well as non-deep learning methods, require two separate stages for graph related applications (graph representation generation and then the specific applications) which is sub-optimal, because the learned graph representations might not be suitable for the downstream applications.

2.3 Deep Learning Method

2.3.1 Graph Neural Network based Methods

Two main types of graph neural networks are introduced in this section: Recurrent-based and Convolutional-based graph neural networks:

Recurrent Graph Neural Networks. Graph neural network is firstly proposed in [16] in the form of recurrent GNN, where the node embeddings are generated through iteratively propagating information from neighbor nodes until they reach a stable fixed point [58]. To achieve this state at each epoch, this process becomes computationally expensive. However, it lays the foundation for many following GNN works especially recurrent GNN methods[18, 59, 59–62]. Specifically, SSE[18] proposed to update node hidden states by gathering information from its neighbor nodes and then update network trainable parameters. SSE only contains one layer and this process can be conducted in a mini-batch manner, so SSE is scalable to large graphs while they didn't prove the final node states can reach the stable fix point. Besides, all these methods introduced above are designed for homogeneous information networks and they fail to handle the problem of incompatibility in HINs.

Convolutional Graph Neural Networks. Convolutional graph neural networks are proposed from spectral domain[17, 63] and spatial domain[25, 64] respectively. Specifically, [63] defines the convolution of the graph in the Fourier domain which needs to calculate the eigendecomposition of the graph Laplacian. This method is computationally expensive and the filters they defined are non-spatially localized. Later, [65] introduces Chebyshev expansion of the graph Laplacian to avoid computing eigenvectors and [17] proposes to do convolution within 1-step neighbor nodes to reduce the complexity. From the spatial perspective[25, 66], it is proposed to generate the node representations by aggregating and then updating information from the neighbor nodes of each node at each layer, which is also referred to as message-passing-based structure. By stacking multiple such layers, the receptive field of each node becomes relatively large and the node representations will contain richer information. Both of these two perspectives perform well in node representation learning and based on which, many variants[64, 67, 68] are developed to improve the GNN's performance. Actually, most of the message-passing-based

GNN variants are developed by designing better aggregation and updating functions (Figure 2.1). The good performance of GNNs in node representation learning also inspires people to leverage node embeddings extracted from GNNs to generate graph embeddings for graph-based applications [69, 70]. However, compared with random-walk-based methods [13, 14], GNN-based methods usually fail to capture long-term relationships between nodes which is usually critical in node classification tasks. In addition, to achieve satisfying performance, GNN-based methods [71] usually require suitable initialization which makes GNN-based methods to be impractical in some applications.

FIGURE 2.1: Variants of message-passing-based GNN model.

There are also many methods designed based on convolutional-based graph neural networks for HINs [20, 23, 72, 73]. Specifically, HGT [72] applies node-level attention strategy at each GNN layer when aggregating information from different types of neighbor nodes, and they also proposed a mini-batch sampling strategy for HIN. Different from HGT, ie-HGCN [20] only applied node-type-level attention strategy when aggregating information from neighbor nodes which further improved the training efficiency. However, for both HGT and ie-HGCN, the length of possible meta-paths is limited by the number of GNN layers. HAN [73] is also designed based on meta-paths where each meta-path is processed with a GAT and the final node representations are generated by combining the results of these GATs. Similar methods include HERec [4], ActiveHNE [74] and MAGNN [23]. Specifically, MAGNN is an extension of HAN except that the information of intermediate nodes along meta-paths is also considered. However, these methods still require manually setting meta-paths in advance.

In GNN-based methods, when it comes to graph-level representation learning, the key becomes designing an effective pooling module to generate graph representation based on the well-learned node representations [66, 69, 75, 76]. The most

straightforward method is mean-pooling which generates the graph representation as the average of all node representations and it is applied in DCNN[75]. In DGCNN[66], node representations are firstly generated by concatenating the node representations generated from all layers, then these nodes are ordered based on the node representations extracted from the last layer of the GNN (sort-pooling). Lastly, the 1-D CNN is applied for graph classification. GCAPS-CNN[76] introduces covariance-based pooling where the covariance value between all nodes at each channel is calculated and all the covariance values will be reshaped to the graph representation. However, all the methods introduced above encode both node and graph representations in the manner of the scalar which is inefficient and more analysis can be found in Chapter 4.

2.3.2 Others

Inspired by the main idea that the procedure of embedding nodes into lower dimensional space can be considered as dimensionality reduction. Autoencoder, one of the most widely used non-linear dimensionality reduction algorithms, is naturally applied to learn the node representations[77–79]. To be more specific, the input and the output of the autoencoder are set as the adjacency matrix or related matrix (e.g., PPMI), and the features extracted at the middle layer of the autoencoder are considered as the node representations. However, these neural network based auto-encoders have difficulties in handling large sparse networks which might lead to expensive computation cost and sub-optimal performance.

Besides, there are some algorithms[80–83] that are designed mainly based on other well-developed deep-learning algorithms. For example, PSCN[84] first sorts all nodes of a graph, then defines a receptive-field size for each node. These receptive-field are then filled with sorted neighbor nodes. Lastly, 1-D convolutional neural network(CNN)[85] is applied to perform graph classification. However, finding universally applicable criteria to sort nodes is challenging. So there is a limited number of works in this category. HeGAN[71] and GraphGAN[86] make use of generative model for graph representation learning which, however, highly depends on the initialization where pre-training is required.

It is noted that, normally, the GNN-based methods refer to a group of neural network methods designed for specific graph-related applications and can be trained in

a data-driven manner. However, by introducing the concept of proximity preserving from graph embedding based methods, some GNN-based methods[[17](#), [23](#), [25](#), [72](#)] can also be trained through unsupervised learning. However, suitable initialization is still critical to its performance of this category of methods.

Chapter 3

Preliminaries

To simplify the reading of the following chapters, the basic notations used for describing different types of graphs are given in this chapter. Besides, common experimental datasets that are widely used in graph-related tasks are also introduced.

3.1 Notation and Definition

The definition and corresponding notations used for describing normal graphs are introduced firstly which represents the most general case, based on which, we introduce the definitions and corresponding notations used for describing HINs as well as their related concepts. HIN is especially detailed introduced here because it is more practical and common in many applications and it is also the focus of some of our works.

FIGURE 3.1: A example of normal graph.

FIGURE 3.2: A simple example of HIN (IMDB information network) and the corresponding network schema.

Normal Graph. By definition, a directed normal graph can be represented by $\mathcal{G} = (\mathbb{V}, \mathbb{E}, \mathbf{X})$ where $\mathbb{V} = \{n_1, n_2, \dots\}$ and $\mathbb{E} = \{e_1, e_2, \dots\}$ represent the sets of nodes and edges respectively. The edges of a graph sometimes are also represented with an adjacency matrix $\mathbf{A} \in \{0, 1\}^{N \times N}$. If there is an edge from n_j to n_i , then $\mathbf{A}_{ij} = 1$ otherwise $\mathbf{A}_{ij} = 0$. $\mathbf{X} \in \mathbb{R}^{N \times d}$ represents the features of each node where d is the number of feature channels and N is the total number of nodes. A simple example can be found in Figure 3.1.

Heterogeneous Information Network (HIN). Same as the normal graph, a heterogeneous information network can be also denoted as $\mathcal{G} = (\mathbb{V}, \mathbb{E}, \mathbf{X})$. In addition to the basic information, the heterogeneous information network is also associated with a node type mapping $\phi : \mathbb{V} \rightarrow \mathbb{T}^v$ and an edge type mapping $\psi : \mathbb{E} \rightarrow \mathbb{T}^e$ where $\mathbb{T}^v$ and $\mathbb{T}^e$ denote the sets of node and edge types respectively. In a HIN, $|\mathbb{T}^v| + |\mathbb{T}^e| > 2$. Together with HIN, there is another important concept named **Network Schema** that needs to be introduced. Denoted as $\mathcal{T}_{\mathcal{G}} = (\mathbb{T}^v, \mathbb{T}^e)$, network schema is a meta template for the HIN $\mathcal{G}$. It is a directed graph defined over node types $\mathbb{T}^v$ with edges as relations from $\mathbb{T}^e$. Network schema defines the type constraints on the sets of nodes as well as the allowed relationships between these nodes.

Figure 3.2a illustrates a simple example of HIN with five different types of objects (User(U), Movie(M), Actor(A), Genre(G) and Director(D)) and four different

types of relationships (user reviewed movie(U-M), actor played movie(A-M), director directed movie(D-M), genre of movie(G-M)). Figure 3.2b displays the network schema of the HIN showed in Figure 3.2a in which nodes U, M, A, G, D represent the object type of User, Movie, Actor, Genre, and Director respectively. Different types of relationships between different types of objects are also displayed.

3.2 Common Datasets

In this section, we introduce some of the common datasets that are widely used in both graph-level and node-level representation learning tasks. The corresponding pre-processing details are also introduced.

Graph-level. As for graph-level representation learning, five biological graph datasets: MUTAG, ENZYMES, NCI1, PROTEINS, D&D, and five social network datasets: COLLAB, IMDB-B, IMDB-M, RE-M5K, RE-M12K [52] are usually used for experimental study. Details of these datasets can be found in Table 3.1.

TABLE 3.1: Description of Datasets for Graph-level Representation Learning

Dataset	Source	Graphs	Classes	Nodes Avg.	Edges Avg.	Nodes Labels
MUTAG	Bio	188	2	17.93	19.79	7
ENZYMES	Bio	600	6	32.46	63.14	6
NCI1	Bio	4110	2	29.87	32.30	23
PROTEINS	Bio	1113	2	39.06	72.81	4
D& D	Bio	1178	2	284.31	715.65	82
COLLAB	Social	5000	3	74.49	4914.99	-
IMDB-B	Social	1000	2	19.77	193.06	-
IMDB-M	Social	1500	3	13	131.87	-
REDDIT-M5K	Social	4999	5	508.5	1189.74	-
REDDIT-M12K	Social	11929	11	391.4	456.89	-

Node-level. The datasets that are widely used for node-level representation learning tasks are summarized in Table 3.2. Specifically, **DBLP** and **CORA**¹ are bibliographical networks. DBLP used here is DBLP-four-areas [87]. It is extracted from DBLP database which contains papers(P) that are published in 20 conferences(V)² as well as the related authors(A) and the key words(T) of these papers. As for

¹<https://people.cs.umass.edu/~mccallum/code-data.html>

²IJCAI, AAAI, ICML, CVPR, ECML, ICDE, VLDB, PODS, EDBT, SIGMOD, SIGIR, CIKM, WWW, ECIR, WSDM, KDD, PAKDD, ICDM, PKDD, SDM.

TABLE 3.2: Statistics of The Experimental Datasets

Dataset	Edge Type	Node Type	Num. of Nodes	Feature	Num. of Labels
Douban Movie 1	$U \rightleftharpoons U$,	G	2,753	—	—
	$U \rightleftharpoons G$,	U	13,367	—	—
	$U \rightleftharpoons M$,	M	12,677	—	36
	$M \rightleftharpoons A$,	A	6,311	—	—
	$M \rightleftharpoons D$	D	2,449	—	—
Douban Movie 2	$U \rightleftharpoons U$,	G	2,269	—	—
	$U \rightleftharpoons G$,	U	3,022	—	—
	$U \rightleftharpoons M$,	M	6,971	—	36
	$M \rightleftharpoons A$,	A	3,004	—	—
	$M \rightleftharpoons D$	D	789	—	—
IMDB	$U \rightleftharpoons M$,	A	42,275	—	—
	$D \rightleftharpoons M$,	M	1,360	—	—
	$A \rightleftharpoons M$,	U	943	—	—
	$G \rightleftharpoons M$,	G	23	—	—
		D	918	—	—
IMDB-L	$A \rightleftharpoons M$,	A	3,836	—	—
	$D \rightleftharpoons M$,	M	8,252	—	—
	$U \rightleftharpoons M$,	D	453	—	—
	$T \rightleftharpoons M$,	U	2,028	—	—
		T	257	—	—
IMDB-S	$A \rightleftharpoons M$,	A	5,257	3,066	—
	$D \rightleftharpoons M$,	M	4,278	3,066	3
		D	2,081	3,066	—
CORA	$A \rightleftharpoons P$,	A	17,411	—	—
	$P \rightleftharpoons P$,	P	19,396	11,119	10
	$P \rightleftharpoons T$	T	12,313	100	—
DBLP	$A \rightleftharpoons P$,	A	14,475	—	—
	$V \rightleftharpoons P$,	P	14,376	—	4
	$T \rightleftharpoons P$,	T	8,920	—	—
		V	20	—	—
Last_FM	$A \rightleftharpoons T$,	A	17,632	—	—
	$A \rightleftharpoons U$,	U	1,892	—	—
	$U \rightleftharpoons U$	T	1,088	—	—
Yelp	$Ca \rightleftharpoons B$,	Ca	1,020	—	—
	$Ci \rightleftharpoons B$,	Ci	10	—	—
	$U \rightleftharpoons B$,	U	206,340	—	—
		B	117,285	—	10

CORA, there are only 3 types of nodes: authors(A), papers(P), and Terms(T) where paper nodes are labeled based on their research areas and there are 10 research areas³ in all. In CORA, each paper node is described by the TF-IDF representation of its paper title and we use GloVe [88] word embeddings to describe each term node. **IMDB** [49] links the movie-attribute information from IMDB and the user-reviewing information from MovieLens100K [89] to form HIN and there are 5 types of nodes (Movie(M), User(U), Director(D), Genre(G), Actor(A))

³Information Retrieval, Databases, Artificial Intelligence, Encryption and Compression, Operating Systems, Networking, Hardware and Architecture, Programming, Data Structures Algorithms and Theory, Human Computer Interaction.

in IMDB. **IMDB-L** is the subsets of IMDB used in HetRec2011⁴ which is originally designed for recommendation. Similar to IMDB-L, **IMDB-S**⁵ [23] is also a subset of IMDB while it is scraped from online. Another difference is that each movie node in IMDB-S is described with the bag-of-words representations of its plot keywords which provides more representative information. **Douban Movie 1**⁶ and **Douban Movie 2**⁶ [90] are also movie-related datasets which are scraped from Douban social media network and it contains 5 types of nodes (Movie(M), User(U), Director(D), Genre(G), Actor(A)). Different from IMDB series datasets, Douban Movie also provide users interaction information. **Yelp** is a social media dataset and it is released in Yelp Dataset Challenge. We extract the data of 10 top cities⁷ with the most businesses(B) to form the HIN and 4 types of nodes (business(B), user(U), Category of business(Ca), City of business(Ci)) are included in this dataset. Following HIN2vec [28], in Yelp, we select 10 main cuisines⁸ in restaurants' categories as labels and there are 17,782 restaurant business in all can be labeled with one of the 10 cuisine types. **Last_FM**⁵ is the subsets of Last.FM used in HetRec2011⁹ which is also designed for recommendation.

⁴<https://grouplens.org/datasets/hetrec-2011/>

⁵<https://github.com/cynricfu/MAGNN>

⁶http://shichuan.org/HIN_dataset.html

⁷the 10 top cities are 'Las Vegas', 'Toronto', 'Phoenix', 'Charlotte', 'Scottsdale', 'Calgary', 'Pittsburgh', 'Montreal', 'Mesa', 'Henderson'

⁸American, French, Italian, Mexican, Canadian, Thai, Indian, Japanese, Chinese, Vietnamese

⁹<https://grouplens.org/datasets/hetrec-2011/>

Chapter 4

Capsule Graph Neural Network

4.1 Introduction

This chapter mainly focuses on the roughest level: graph-level representation learning, where the main challenge is the pooling process, the purpose of which is to generate graph representations by appropriately combining information extracted from node representations.

The success of GNN in node representation learning has inspired many deep-learning-based approaches to leverage on node embeddings extracted from GNN to generate graph embeddings for graph-based applications. However, during this procedure, the learned distributed representation of each node will be considered as multiple individual scalar features instead of one vector. For example, DGCNN[66] applies element-wise max-pooling to nodes embeddings during the procedure of generating graph embeddings, GCaps-CNN[76] generates graph embeddings by computing the element-wise covariance of all nodes. These operations indicate that the node features are extracted in the form of scalar when they are used to generate graph embeddings and this is sub-optimal when considering representing efficiency.

To build high-quality graph embeddings, it is important to not only detect the presence of different structures around each node but also preserve the corresponding detail properties such as node position, structure direction and connection, etc. However, encoding these properties information in the form of scalar means

activating elements in a vector one-by-one which is exponentially less efficient than encoding them with distributed representations and this has been identified discussed in CapsNet[19]. So, inspired by CapsNet, we propose to extend scalar to vector during the procedure of applying GNN to graph representation learning. Since compared with scalar-based neural network, vector-based neural network preserves the information of node/graph properties more efficiently.

In this chapter, we present Capsule Graph Neural Network (CapsGNN), a novel deep learning architecture, which is inspired by CapsNet and uses node features extracted from GNN to generate high-quality graph embeddings. More specifically, basic node features are extracted in the form of capsules through GNN and routing mechanism is applied to generate high-level graph capsules as well as class capsules. In the procedure of generating graph capsules, an Attention Module can be applied to tackle graphs in various sizes. It also assigns different weights to each node capsules so that this model focuses on more critical parts of the graph. Our main contribution are listed below:

- We proposed a novel message-passing-based architecture(CapsGNN) for direct graph processing where the capsule technique is well integrated to improve the representing efficiency. Multiple graph embeddings will be generated for each graph to separately reflect different properties of the graph. To the best of our knowledge, we are the first to introduce the concept of capsules to graph representation learning, by applying routing mechanism, extracted graph features are generated in the form of distributed representations.
- We validate the performance of generated graph embeddings on classification task over 5 biological datasets and 5 social datasets. CapsGNN achieves state-of-the-art performance on 6 out of 10 benchmark datasets and comparable results on the rest.
- T-SNE[30] is used to visualize the learned graph embeddings and the results show that different graph capsules indeed capture different information of the graphs.

4.2 Preliminary and Related Work

Here, we provide a brief introduction to graph classification, details of GNN, routing mechanism proposed from CapsNet and Attention mechanism which is used in CapsGNN.

4.2.1 Problem statement

Given a set of attributed graphs $\mathbb{G} = \{(\mathcal{G}_1, l_1), (\mathcal{G}_2, l_2), \dots, (\mathcal{G}_n, l_n)\}$, we intend to learn a function $f : \mathbb{G} \rightarrow \mathcal{L}$, where $\mathbb{G}$ is the input set of graphs and $\mathcal{L}$ is the corresponding set of graph labels. More details of the notations used for graph description can be found in Section 3.1.

4.2.2 Graph Neural Network

Here we focus on introducing Graph Convolutional Networks (GCNs) [17], a widely used GNN architecture, which is also one of the key building blocks in this work. At each layer of the GCN, the convolution operation is applied to each node as well as its neighbors and the new representation of each node is computed through an activation function. This procedure can be written as:

$$\mathbf{Z}^{(l+1)} = f(\mathbf{T}\mathbf{Z}^l\mathbf{W}^l) \quad (4.1)$$

where $\mathbf{Z}^l \in \mathbb{R}^{N \times d}$ represents node features at the layer l , d represents the number of feature channels and $\mathbf{Z}^0 = \mathbf{X}$, $\mathbf{W}^l \in \mathbb{R}^{d \times d'}$ is a trainable weights matrix which serves as a channel filter, $f(\cdot)$ is a nonlinear activation function, $\mathbf{T} \in \mathbb{R}^{N \times N}$ is the information transform matrix which is usually calculated from the adjacency matrix $\mathbf{A}$ for guiding the information passing among nodes.

A complete GNN usually stacks L layers to generate the final node embeddings $\mathbf{Z}^L$. In the architecture proposed by GCN [17], at the l th layer of GNN, the extracted features of each node actually take all its adjacent nodes within l steps into consideration. So l can be considered as the size of the node receptive-field at this layer. This special property inspired us to use node features extracted from different layers to generate the higher-level graph capsules.

4.2.3 Capsule Neural Network

The technique for extracting features in the form of vectors is proposed in [29] and improved in [19] and [91]. This technique is mainly devised for image processing domain. In their work, the extracted vector is referred to as capsule (a group of neurons in neural network), so we follow the same notation in this work. Introducing capsules allows us to use routing mechanism to generate high-level features which we believe is a more efficient way for features encoding. Compared with max-pooling in CNN in which all information will be dropped except for the most active one, routing preserves all the information from low-level capsules and routes them to the closest high-level capsules. More specifically, in CapsNet [19], the direction of capsules reflects the properties of the features and the length of capsules reflects the probability of the presence of different features. The transmission of information between layers follows Dynamic Routing mechanism and the specific procedure of Dynamic Routing is shown Algorithm 1:

Algorithm 1 Dynamic routing mechanism returns parent capsules $\mathbf{H}$ given children capsules $\mathbf{S}$, a set of trainable transform matrices $\mathbb{W}$ and the number of iterations t .

```

1: procedure DYNAMIC ROUTING( $t, \mathbf{S}, \mathbb{W}$ )
2:   for all children capsule  $i$  :  $\vec{v}_{j|i} = \vec{s}_i^T \mathbf{W}_{ij}$ 
3:   for all children capsule  $i$  to all parent capsule  $j$ :  $r_{ij} \leftarrow 0$ 
4:   for  $t$  iterations do
5:     for all children capsule  $i$ :  $\tilde{r}_i \leftarrow \text{softmax}(\vec{r}_i)$ 
6:     for all parent capsule  $j$ :  $\tilde{h}_j \leftarrow \sum_i \tilde{r}_{ij} \vec{v}_{j|i}$ 
7:     for all parent capsule  $j$ :  $\tilde{h}_j \leftarrow \text{squash}(\tilde{h}_j)$ 
8:     for all children capsule  $i$  to all parent capsule  $j$ :  $r_{ij} \leftarrow r_{ij} + \tilde{h}_j^T \vec{v}_{j|i}$ 
9:   end for
10:  return  $\tilde{h}_j$ 
11: end procedure

```

Inspired by CapsNet, the capsule mechanism is adopted and fused with GNN in our proposed CapsGNN to generate graph capsules and class capsules on the basis of node capsules which are extracted from GNN. Dynamic Routing is applied to update weights between capsules from one layer to the next layer so that the properties captured by node capsules can be propagated to the suitable graph capsules. Thus, each graph is modeled as multiple graph capsules, and then modeled as multiple class capsules. Different graph capsules reflect the properties of the graph from different aspects.

4.2.4 Attention Mechanism

Attention mechanism is widely applied in image processing [84] and natural language processing domain [92] where it is used to find the relevant parts of the input data to the task target. The main procedure of Attention mechanism is:

- 1) Defining an attention measure which is used to measure the relevance of each part of the input data to the task target;
- 2) Normalizing the generated attention value;
- 3) scaling each part with the normalized attention value.

In CapsGNN, we apply Attention mechanism for two purposes:

- 1) scaling node capsules so that the generated graph capsules from different graphs are still comparable even though these graphs are vastly different in sizes.
- 2) guiding the model to focus on more relevant and important parts of graphs.

4.3 Methodology

In this section, we outline CapsGNN and show how it is used to generate high-quality graph capsules which then can be applied to graph classification task.

4.3.1 Overall Architecture

Figure 4.1 shows a simplified version of CapsGNN. It consists of three key blocks:

- 1) **Basic node capsules extraction block:** GNN is applied to extract local vertices features with different receptive-field and then primary node capsules are built in this block.
- 2) **High level graph capsules extraction block:** Attention Module and Dynamic Routing are fused to generate multiple capsules for graphs.

FIGURE 4.1: Framework of CapsGNN.

- 3) **Graph classification block:** Dynamic Routing is applied again to generate final graph class capsules for graph classification.

The details of each block is explained in the following.

4.3.2 Basic Node Capsules

Firstly, the basic node features are extracted with GNN. Node degrees are used as node attributes if nodes do not have attributes. We use the architecture improved by GCN [17] as the node features extractor. The difference is that we extract multi-scale node features from different layers and the extracted features are represented in the form of capsules. The procedure can be written as:

$$\mathbf{Z}_j^{l+1} = f\left(\sum_i \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{Z}_i^l \mathbf{W}_{ij}^l\right) \quad (4.2)$$

where $\mathbf{W}_{ij}^l \in \mathbb{R}^{d \times d'}$ is the trainable weights matrix. It serves as the channel filters from the i th channel at the l th layer to the j th channel at the $(l+1)$ th layer. Here, we choose $f(\cdot) = \tanh(\cdot)$ as the activation function. $\mathbf{Z}^{l+1} \in \mathbb{R}^{N \times d'}$, $\mathbf{Z}^0 = \mathbf{X}$, $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ ¹ and $\tilde{\mathbf{D}} = \sum_j \tilde{\mathbf{A}}_{ij}$. To preserve features of sub-components with different sizes, we use node features extracted from all GNN layers as well as the input layer to generate high-level graph capsules.

¹ $\mathbf{I}$ represents identity matrix.

4.3.3 High-Level Graph Capsules

After getting local node capsules, global routing mechanism is applied to generate graph capsules. More specifically, the input of this block contains N sets of node capsules and each set is $\mathbb{S}^n = \{\vec{s}_{1,1}, \dots, \vec{s}_{1,C_1}, \dots, \vec{s}_{L,C_L}\}$, $\vec{s}_{l,c} \in \mathbb{R}^d$, where C_l is the number of channels at the l th layer of GNN, d is the capsule dimension. The output of this block is a set of graph capsules $\mathbf{H} \in \mathbb{R}^{P \times d'}$. Each of the capsules reflects the properties of the graph from different aspects. The length of these capsules reflects the probability of the presence of these properties and the angle reflects the details of the graph properties. Before generating graph capsules with node capsules, an Attention Module is introduced to scale node capsules.

Attention Module. In CapsGNN, primary capsules are extracted based on each node which means the number of primary capsules depends on the size of input graphs. In this case, if the routing mechanism is directly applied, the value of generated high-level capsules will highly depend on the number of primary capsules (graph size) which is not the ideal case. Hence, an Attention Module is introduced to combat this issue.

The attention measure we choose to use is a two-layer fully connected neural network $F_{attn}(\cdot)$. The number of the input units of $F_{attn}(\cdot)$ is $d \times C_{all}$ where $C_{all} = \sum_l C_l$ and the number of the output units equals to C_{all} . We apply node-based normalization to generate attention value in each channel and then scale the original node capsules. The details of Attention Module is shown in Figure 4.2. Specifically, we first flatten primary capsules and apply two layer fully-connected neural network to generate attention value for each capsule. Then node-based normalization (normalize each row here) is applied to generate final attention value. At last, scaled capsules are calculated by multiplying the normalized value with primary capsules. This procedure can be written as:

$$scaled(\vec{s}_{(n,i)}) = \frac{F_{attn}(\tilde{\vec{s}}_n)_i}{\sum_n F_{attn}(\tilde{\vec{s}}_n)_i} \vec{s}_{(n,i)} \quad (4.3)$$

where $\tilde{\vec{s}}_n \in \mathbb{R}^{1 \times d C_{all}}$ is obtained by concatenating all capsules of the node n . $\vec{s}_{(n,i)} \in \mathbb{R}^{1 \times d}$ represents the i th capsule of the node n and $F_{attn}(\tilde{\vec{s}}_n) \in \mathbb{R}^{1 \times C_{all}}$ is the generated attention value. In this way, the generated graph capsules can be

independent to the size of graphs and the architecture will focus more on important parts of an input graph.

FIGURE 4.2: The structure of Attention Module in CapsGNN.

Coordinate Addition. Coordinate addition module can be used to preserve the position information of each node during the procedure of generating node capsule votes after the attention module. Here, we introduce coordinate addition as an additional module which might be helpful to improve the quality of graph representations in some datasets. The details of Coordinate Addition module can be found in Figure 4.3. It is noted that this module is not necessary in some datasets and we propose this module just as an optimization module. When the GNN goes deeper, the extracted nodes features contain more specific position information of each node. Inspired by DGCNN [66] where the node embeddings learned from the last layer of GNN are taken to order all nodes, we also take the capsules extracted from the last layer of GNN as the position indicators of corresponding nodes by concatenating it with each capsule of the node. The procedure of calculating votes with node position indicators can be written as:

$$\vec{v}_{(n,i),j} = [\vec{s}_{(n,i)}^T \mathbf{W}_{ij}^n \| \vec{s}_{(n,C_{all})}^T \mathbf{W}_j^p] \quad (4.4)$$

where $\vec{v}_{(n,i),j} \in \mathbb{R}^{1 \times (d_n + d_p)}$ represents the node capsule vote from the i th channel of the n th node to the j th channel of parent capsules. $\mathbf{W}_{ij}^n \in \mathbb{R}^{d \times d_n}$ and $\mathbf{W}_j^p \in \mathbb{R}^{d \times d_p}$ are the transform matrix. $\vec{s}_{(n,i)}$ is the same as introduced in **Attention Module** and $\|$ represents concatenate operation.

FIGURE 4.3: The structure of Coordinate Addition Module.

Graph Capsules Generation. The procedure of generating multiple graph capsules is summarized as follows:

- 1) Scale primary capsules: Apply Attention Module to scale primary capsules. The results of this module should be $\mathbf{S} \in \mathbb{R}^{N \times C_{all} \times d}$.
- 2) Calculate votes: When calculating votes, capsules of different nodes from the same channel share the transform matrix. The result of this step is a set of votes $\mathbf{V} \in \mathbb{R}^{N \times (C_{all}) \times P \times d}$ where C_{all} denotes the number of channels. P denotes the defined number of graph capsules.
- 3) Dynamic Routing Mechanism: High-level graph capsules are computed with the procedure introduced in Section 4.2.3 based on votes produced in previous steps.

4.3.4 Classification

This block is designed for graph classification using the graph capsules.

Classification Loss. Dynamic Routing is applied again over graph capsules to generate final class capsules $\mathbf{C} \in \mathbb{R}^{K \times d}$, where K is the number of graph classes.

FIGURE 4.4: The structure of Reconstruction Loss Module.

Here, we use margin loss function proposed in CapsNet [19] to calculate the classification loss and it is computed as:

$$Loss_c = \sum_k \{T_k \max(0, m^+ - \|\vec{c}_k\|)^2 + \lambda(1 - T_k) \max(0, \|\vec{c}_k\| - m^-)^2\} \quad (4.5)$$

where $m^+ = 0.9$, $m^- = 0.1$ and $T_k = 1$ iff the input graph belongs to class k . λ is used to stop initial learning from reducing the length of all class capsules especially when K is large.

Reconstruction Loss. We use reconstruction loss as regularization method. Here, all class capsules are masked except the correct one and it is decoded with two fully-connected layer to reconstruct the input information. The information we reconstruct here is the histogram of input nodes. The procedure can be written as:

$$Loss_r = \frac{\sum_i MP_i (\vec{d}_i - \vec{m}_i)^2}{\sum_i MP_i} + \frac{\sum_i (1 - MP_i) (\vec{d}_i - \vec{m}_i)^2}{\sum_i (1 - MP_i)} \quad (4.6)$$

where $\vec{m}_i$ represents the number of nodes with the attribute i appear in the input graph, $\vec{d}_i$ is the corresponding decoded value. $MP_i = 1$ iff input graph contains nodes with attribute i . Equation (4.6) is designed in this way to prevent reducing reconstruction loss from setting all decoded value as 0 especially when most of the elements of the ground truth are 0. The structure of reconstruction loss module is shown in Figure 4.4.

4.4 Differences from Existing Approaches

The architecture details presented in Section 4.3.1 described the key design idea of CapsGNN which is based on the fusing of GNN and CapsNet. In this section, we present a general comparison between CapsGNN with existing approaches:

1. Compared with DCNN [75], ECC [93] and DGCNN [66] (GNN-based graph representation learning architectures), CapsGNN represents node features in the form of capsules. This is helpful to preserve the properties information contained in nodes more efficiently when generating graph embeddings. Besides, each graph is modeled as multiple embeddings in CapsGNN instead of only one embedding used in other approaches. This allows us to capture information of graphs from different aspects. The second difference is that, in these approaches, each part of the graph is given equal importance. However, the attention mechanism used in CapsGNN allows it to assign various weighting to different nodes. This leads the model to focus more on critical parts of input graphs. Lastly, different from DCNN [75] and ECC [93], CapsGNN and DGCNN [66] uses node features extracted from multiple layers of GNNs so that different size of receptive-fields are applied to preserve more information.
2. GCAPS-CNN [76] also introduced capsules-related concept into graph representation learning. However, they generate capsules in a feature-based manner instead of learning capsules as distributed embeddings. More specifically, when they extend a scalar feature to a capsule for the node n , P higher-order statistical moment value is calculated based on its neighbor nodes and this P value is concatenated to a P -dimensional capsule. Between layers, GCAPS-CNN performs dimension reduction to compress capsules back to scalar features, which defeats the purpose of having capsules in the first place. CapsGNN learns each dimension of capsules in a data-driven manner and apply routing mechanism between layers to preserve the learned meaning of each capsule. This also allows us to preserve multiple properties information contained in nodes more efficiently especially when generating graph embeddings.

3. Compared with CapsNet [19] which works well in image processing domain, CapsGNN needs to handle more complex situations in graphs. In image processing domain, the size of the input images can be standardized by resizing the images. However, it is not possible to simply resize the graphs. So, we introduced an additional Attention Module to tackle graphs that are vastly different in sizes and preserve important parts of graphs. We also propose to use features extracted from all layers of GNN since it is hard to define a suitable receptive-field size for graphs. Furthermore, compared with the architecture of CapsNet, CapsGNN has one additional graph capsules layer which is used to learn multiple graph embeddings and these embeddings reflect different aspects of graph properties which is valuable in future research. To the best of our knowledge, we are the first one to model a graph as multiple embeddings in the form of distributed capsules and we believe this approach of learning representations has a high potential for other complex data analysis which is not limited to graphs. Besides, CapsGNN has different explanation of linear transformation. In CapsNet, by applying a linear trainable transformation to pose vectors, the spatial relationship between object parts and the whole object can be well modeled. However, by applying linear trainable transformation, CapsGNN is simply computing the prediction vectors from nodes-level representations to graph-level representations. This transform matrix is not trying to model the change of viewpoint or capture viewpoint invariant knowledge but to model the relationship between the nodes and the whole graph.

4.5 Experiments

We verify the performance of graph embeddings learned from CapsGNN against a number of state-of-the-art approaches and some classical approaches on classification task with 10 benchmark datasets. Besides, we conduct experimental study to assess the impact of capsules in efficiency of encoding features of graphs. We also conduct brief analysis on the generated graph/class capsules. The experimental setting, results and analysis is shown in the following. In addition to the analysis of the whole architecture, we also provide a comparison experiment to evaluate the

contribution of each module of CapsGNN in classification task and a brief analysis of learned attention distribution. More details can be found below.

4.5.1 Dataset

Five biological graph datasets: MUTAG, ENZYMES, NCI1, PROTEINS, D&D and five social network datasets: COLLAB, IMDB-B, IMDB-M, RE-M5K, RE-M12K [52] are used for our experimental study. Details of these datasets can be found in Table 3.1.

4.5.2 Baselines Methods

We compare CapsGNN with both kernel-based and deep-learning-based algorithms. The details are given as follows:

Kernel-based Methods: Three kernel-based algorithms, namely the Weisfeiler-Lehman subtree kernel (WL) [9], the graphlet count kernel(GK) [33], and the Random Walk (RW) [10] are selected. Typically, kernel-based algorithms first decompose graphs into predefined sub-components based on the kernel definition, then build graph embeddings in a feature-based manner. Lastly, some machine learning algorithms (i.e., SVM) are applied to perform graph classification.

Deep-Learning-based Methods: Three types of deep-learning-based algorithms are selected:

- 1) Graph2vec [55], Deep Graph Kernel (DGK) [52] and AWE [56]: Graph2vec, DGK and AWE require extracting sub-structures in advance while Graph2vec and AWE learn the representation of graphs in the manner of Doc2vec [54], DGK applies Word2vec [37] to learn the similarity between each pair of sub-structures which will be used to build graph kernel. Then kernel-based machine learning methods (i.e., SVM) are applied to perform graph classification. These three algorithms as well as kernel-based methods are all sub-components based which require computationally intensive preprocessing effort. So although Graph2vec, DGK and AWE apply learning approaches to learn the embeddings, we still consider them and other kernel-based algorithms as the same type in

our experiments and we mainly compare our proposed architecture with the other remained methods which are all end-to-end and totally data-driven architectures.

- 2) PATCHY-SAN (PSCN) [80]. This method first sorts all nodes, then defines a receptive-field size for each node. These receptive-field are then filled with sorted neighbor nodes. Lastly, 1-D CNN is applied to perform graph classification.
- 3) GCAPS-CNN [76], Dynamic Edge CNN (ECC) [93] and Deep Graph CNN (DGCNN) [66]. These methods are all GNN-based algorithms. GCAPS-CNN first extract FGSD [94] features for nodes without attributes and then generate capsules for each node with higher-order statistical moment value of its neighbor nodes. At the last layer, they calculate covariance between all nodes to generate graph embeddings. ECC extracts node features on the condition of edge labels in GNN and then apply multi-scale pyramid structure to coarsen the graph. It uses average pooling at the last layer to generate graph embeddings. DGCNN generates nodes embeddings through a multi-layer GNN and combine features extracted from all layers. Then they order the nodes based on the embeddings extracted from the last layer which is followed by 1-D CNN for classification.

4.5.3 Experimental Set-up

The same architecture setting is used in CapsGNN for all datasets to show its robust performance. For the node capsules extraction, the GNN has 5 layers ($L = 5$), the number of channels at each layer is set as the same which is 2 ($C_l = 2$). The number of graph capsules is fixed as 16 ($P = 16$). The dimension of all capsules are set as 8 ($d = 8$). The number of units in the hidden layer of Attention Module is set as $\frac{1}{16}$ of the number of input units. The number of iterations in routing is set as 3. During training stage, we simultaneously reduce $Loss_c$ and $Loss_r$ and we scale $Loss_r$ with 0.01 so that the model focuses on classification task. λ is set as 0.5 and 1.0 for multi-class classification and binary classification respectively.

To evaluate the performance objectively, we applied 10-fold cross validation on each dataset. Each time we use 1 training fold as validation fold to adjust hyper-parameters, 8 training fold to train weights and the remained 1 testing fold to test the performance. We stop training when the performance on validate fold start to

TABLE 4.1: Experiment Result of Biological Dataset

Algorithm	MUTAG	NCI1	PROTEINS	D&D	ENZYMES
WL	82.05±0.36	82.19±0.18	74.68±0.49	79.78±0.36	52.22±1.26
GK	81.58±2.11	62.49±0.27	71.67±0.55	78.45±0.26	32.70±1.20
RW	79.17±2.07	>3days	74.22±0.42	>3days	24.16±1.64
Graph2vec	83.15±9.25	73.22±1.81	73.30±2.05	-	-
AWE	87.87±9.76	-	-	71.51±4.02	35.77±5.93
DGK	87.44±2.72	80.31±0.46	75.68±0.54	73.50±1.01	53.43±0.91
PSCN	88.95±4.37	76.34±1.68	75.00±2.51	76.27±2.64	-
DGCNN	85.83±1.66	74.44±0.47	75.54±0.94	79.37±0.94	51.00±7.29
ECC	76.11	76.82	-	72.54	45.67
GCAPS-CNN	-	82.72±2.38	76.40±4.17	77.62±4.99	61.83±5.39
CapsGNN	86.67±6.88	78.35±1.55	76.28±3.63	75.38±4.17	54.67±5.67

TABLE 4.2: Experiment Result of Social Dataset

Algorithm	COLLAB	IMDB-B	IMDB-M	RE-M5K	RE-M12K
WL	79.02±1.77	73.40±4.63	49.33±4.75	49.44±2.36	38.18±1.30
GK	72.84±0.28	65.87±0.98	43.89±0.38	41.01±0.17	31.82±0.08
DGK	73.09±0.25	66.96±0.56	44.55±0.52	41.27±0.18	32.22±0.10
AWE	73.93±1.94	74.45±5.83	51.54±3.61	50.46±1.91	39.20±2.09
PSCN	72.60±2.15	71.00±2.29	45.23±2.84	49.10±0.70	41.32±0.42
DGCNN	73.76±0.49	70.03±0.86	47.83±0.85	48.70±4.54	-
GCAPS-CNN	77.71±2.51	71.69±3.40	48.50±4.10	50.10±1.72	-
CapsGNN	79.62±0.91	73.10±4.83	50.27±2.65	52.88±1.48	46.62±1.90

reduce. Then we report the accuracy on test fold as our test result. The final result is the average of these 10 test accuracy. By default, we use the results reported in their original work for baseline comparison. However, in cases where the results are not available, we use the best testing results reported in [56, 66, 76].

4.5.4 Classification Result

Table 4.1 lists the results of the experiments on biological datasets, Table 4.2 lists the results of the experiments on social datasets. For each dataset, we highlight the top 2 accuracy in bold. Compared with all the other algorithms, CapsGNN achieves top 2 on 6 out of 10 datasets and achieves comparable results on the other datasets. Compared with all the other end-to-end architectures, CapsGNN achieves top 1 on all the social datasets.

CapsGNN achieves the state-of-the-art performance on social datasets. More specifically, we are able to improve the classification accuracy by a margin of 2.78% and 5.30% on RE-M5K and RE-M12K respectively. This demonstrates that learning features in the form of capsules and modeling a graph to multiple embeddings is beneficial to capture macroscopic properties of graphs which are more important in classifying social networks. These results also consistent with the property of CapsNet, as it focuses more on extracting important information from children capsules by voting. However, applying routing to the whole graph leads higher-level graph capsules to preserve all the information at a graph level and this property is not suitable to give prominence to individual fine structures which might be more important to biological datasets analysis. This results in less robust of CapsGNN on biological datasets. Despite this, the performance of CapsGNN in graph classification task still demonstrates its capability of graph representation especially its high potential of large graph dataset analysis.

4.5.5 Efficiency of Capsules

In this section, we display the experiment that designed for evaluate the efficiency of capsules in feature encoding. More specifically, more efficient in feature encoding here means representing more information with the similar number of neurons. We construct a scalar-based neural network for each CapsGNN and then compare the CapsGNN with its related scalar-based architecture by comparing their training and testing accuracy on graph classification task to demonstrate the efficiency in feature representation. These scalar-based architectures are designed by replacing the graph capsules block and the class capsules block in CapsGNN with fully-connected layers (FC). In this case, the only difference between each pair of CapsGNN and its corresponding scalar-based architecture is that CapsGNN represents features with vectors and uses routing to propagate information between layers while the scalar-based architecture encodes features with scalar values.

In this experiment, the number of layers of GNN is set as 3 ($L = 3$), the number of channels at each layer is all set as 2 ($C_l = 2$). We construct different CapsGNNs by adjusting the dimension of node (d_n) and graph (d_g) capsules and the number of graph capsules (P). The size of FC in scalar-based architectures is adjusted based on the size of CapsGNNs so that they have comparable number of trainable

TABLE 4.3: Details of Tested Architectures in Efficiency Evaluation Experiment

	2-4-2	2-4-4	2-4-8	2-4-16	4-4-2	4-4-4	4-4-8	4-4-16
No. trainable Para. (Caps)	208	367	688	1328	448	704	1216	2240
No. trainable Para. (Scalar)	216	370	692	1336	452	712	1232	2246
Dim Node Feat. (Both)	2	2	2	2	4	4	4	4
Dim Graph emb. (Caps)	2×4	4×4	8×4	16×4	2×4	4×4	8×4	16×4
Dim FC.(Scalar)	12	23	48	92	10	20	40	79

FIGURE 4.5: Comparison of efficiency in features representation. The horizontal axis represents d_n-d_g-P . The vertical axis represents classification accuracy on NCI1.

weights. Other hyper-parameters are the same as Section 4.5.3. The details of the tested architectures are shown in Table 4.3. Besides, NCI1 dataset, which has more than 4000 graphs, is used for testing and the accuracy of NCI1 on various architectures can be found in Figure 4.5.

In Table 4.3 and Figure 4.5, the setting of different architectures is represented as d_n-d_g-P . Here, we choose the simplest setting (2-4-2) as an example: 2-4-2 means that the dimension of node capsules is $d_n = 2$, the dimension of graph and class capsules is $d_g = 4$ and the number of graph capsules P equals to 2. Besides, we set the dimension of FC of its corresponding scalar-based architecture as 12 so that they have comparable number of trainable weights. In this case,

each graph is modeled as 2 4-dimensional graph embeddings in the CapsGNN or 1 12-dimensional graph embedding in its corresponding scalar-based architecture. Both architectures are sub-optimal to represent the whole dataset while CapsGNN can still reach higher accuracy compared with scalar-based architecture.

As we can see from Figure 4.5, the test accuracy of Caps-based architectures (CapsGNN) is higher than the corresponding scalar-based architectures in all settings. For the training accuracy, when the dimension of FC is slightly higher than the dimension of graph embeddings in CapsGNN, CapsGNN can still reach higher accuracy which indicates that CapsGNN is more powerful in representing the whole dataset. When we keep increasing the number of graph capsules in CapsGNN and enlarging the dimension of FC in scalar-based architectures, the difference between the dimension of graph embeddings and the size of FC becomes larger, their training accuracy will be closer. It is noted that the training accuracy of scalar-based architectures is slightly higher than the CapsGNNs when the dimension of FC is about 20% larger than the dimension of graph capsules. In this experiment, we use extremely simple architectures on purpose to simulate the situation where we need to model complex datasets with relatively simple architectures. Since each pair of CapsGNN and its corresponding scalar-based architecture have similar structure and comparable number of trainable weights, the higher training accuracy and testing accuracy of CapsGNN demonstrate its efficiency in feature encoding and its strong capability of generalization.

4.5.6 Graph Capsules Analysis

CapsGNN leverages on capsules idea to get multiple embeddings for each graph so that complex information underlying graphs can be captured more effectively. To explore the properties of the extracted graph capsules and class capsules, we plot the graph distribution based on capsules extracted from different channels with t-SNE and we take Reddit-12K as an example.

We choose to depict the distribution of graphs which are generated from 3 categories, namely *atheism*, *IAmA* and *mildlyinteresting* with capsules extracted from the 1st, 2nd, 11th, 14th channel of graph capsules. As we can see from Table 4.4, different channels of capsules represent different aspects of graph properties. *atheism* and *IAmA* can be discriminated obviously with capsules extracted from the

TABLE 4.4: Visualization of Graph Capsules

Subreddit	Channel 1	Channel 2	Channel 11	Channel 14
atheism & IAmA
atheism & mildly- interesting

TABLE 4.5: Visualization of Class Capsules

Subreddit	atheism	IAmA	mildly- interesting	Concatenate
atheism(g) & IAmA(r) & mildly- interesting(b)

11th and the 14th channels while they are hard to be separated with capsules extracted from the 1st and the 2nd channels. However, *atheism* and *mildlyinteresting* can be discriminated with the capsules extracted from the 1st and the 2nd channels while they are mixed in the 11th and the 14th channels which is opposite to the case of *atheism* and *IAmA*. This phenomenon can also be observed in other multi-class datasets. It is still hard to figure out the specific aspects these capsules focus on. However, compared with scalar-based neural networks, modeling an object with multiple embeddings makes it possible to explore the meaning of each channel which may lead the model to learn more interpretable embeddings in the future.

As we can see from Table 4.5, different class capsules focus on different classification-related graph properties. For example, the capsules that represent *athesism* (first column) can well discriminate *athesism* (red) from the other two types of graphs while *IAmA* (green) and *mildlyinteresting* (blue) are mixed in this channel. The

similar phenomenon can also be found in other class capsules. Besides, when we concatenate the capsules of these three classes together, three types of graphs can be well discriminated with the concatenated capsules which also directly reflect the classification performance. This property is quite different from standard scalar-based architectures where each graph is modeled with only one graph embedding.² By introducing the concept of capsules, the graph and class capsules can not only preserve classification-related properties of each graph (reflected with the length of class capsules) but also other properties information (reflected with the angle of class capsules). The generated class capsules can also be useful in other follow-up work and we leave this to be explored in the future.

4.5.7 Attention Analysis

We conducted node-based normalization in Attention Module to scale different node capsules to lead the model focusing on critical parts of each graph. In this experiment, we plot the attention distribution on REDDIT and IMDB-M dataset to see if this model indeed focuses on critical parts of graphs. Graphs in IMDB-M are all ego-networks derived from movie-collaboration relationship networks in which each node represents an actor/actress and there is a link between two nodes if they appeared in the same movie. These collaboration networks are built based on movies in different genres namely *Romance*, *Sici-Fi* and *Comedy* and the task is to predict where the given graph belongs. As for REDDIT, each graph in REDDIT dataset represents a discussion thread in which each node indicates a user and there is a link between two users if at least one of them responded to another's comment. The task is to predict the subreddit where the given discussion graph belongs.

The visualization results are shown in Figure 4.6 where we use the color and the size of nodes to represents the learned attention of the corresponding nodes. In these figures, nodes with higher attention are darker in color and larger in size. We can find that the model always pays more attention to nodes with high centrality or critical nodes that connect multiple separate sub-components of the graph which makes sense as critical parts of the graph from common sense. Beside, from the

²More details of graph distribution can be found at <https://sites.google.com/view/capsgnn/home> where we provide more figures generated from COLLAB and REDDIT-MULTI-12K. We also plot the graph distribution based on graph embeddings learned from DGCNN as a reference and the figures can be found at <https://sites.google.com/view/capsgnn/embeddings-visualization/collab/class-capsules>.

FIGURE 4.6: The attention distribution on IMDB-M dataset and REDDIT dataset.

results in Table 4.6, we can find that the Attention Module indeed benefits to the classification accuracy in some datasets.

4.5.8 Contribution of Each Module

In addition to evaluate the performance of the whole architecture on classification task, we also provide detailed study to quantify the contributions made by each module in CapsGNN on classification task. The four comparison architectures Set-up is shown as below and the setting of the hyper-parameters are the same as Section 4.5.3.

1. **GNN + Attention + Routing + Reconstruction + Margin Loss (CapsGNN)**: Basic CapsGNN.
2. **GNN + Attention + Coordinate + Routing + Reconstruction (CapsGNN-Coord)**: Basic CapsGNN with Coordinate Addition module.

TABLE 4.6: Comparison of Each Module

Architecture	COLLAB	IMDB-B	PROTEINS	NCI1	D&D
CapsGNN	79.77±1.15	74.42±2.20	77.27±2.58	79.23±1.88	76.16±4.19
CapsGNN -Coord	80.00±1.22	74.81±2.96	76.58±2.42	79.04±1.93	77.41±3.33
CapsGNN -Avg	79.61±0.81	73.29±2.66	76.54±2.98	78.36±1.95	74.44±3.46
CapsGNN -noRout	80.48±0.86	74.11±2.94	77.18±2.94	77.62±1.15	75.28±4.17
CapsGNN -noRecon	80.44±0.88	74.03±2.11	76.69±1.78	78.23±2.41	74.70±3.12
CapsGNN -Avg-noRout	80.15±1.02	74.23±3.47	75.94±2.61	76.25±2.40	73.93±3.56

3. **GNN + Average + Routing + Reconstruction + Margin Loss (CapsGNN-Avg)**: The Attention module in basic CapsGNN is replaced with the Average module.
4. **GNN + Attention + Reconstruction + Margin Loss (CapsGNN-noRout)**: In this architecture, we will set the similarity between all the capsules from one layer to the next layer as the same so that each children capsule will be equally routed to all the parent capsules.
5. **GNN + Attention + Routing + Margin Loss (CapsGNN-noRecon)**: In this architecture, we directly remove the Reconstruction loss module.

The validation accuracy of each architecture is shown in Table Table 4.6 where we highlight the highest and lowest accuracy respectively. As we can see from the Table, IMDB-B and D&D reach better performance with CapsGNN-Coord which indicates the effectiveness of Coordinate Addition Module. However, the performance of social datasets is still comparable across all types of architectures. On the other hand, the performance of biological datasets(NCI1, PROTEINS, D&D) is more sensitive to the introduced modules in each architecture. The highest accuracy of NCI1, PROTEINS is achieved on CapsGNN which indicates the little effectiveness of Coordinate Addition Module in these two datasets. More specifically, the comparison between CapsGNN-Avg-noRout and CapsGNN-Avg on NCI1, PROTEINS and D&D indicates the effectiveness of Routing mechanism which improves the accuracy by 2.1%, 0.6% and 0.51% respectively. Besides, the comparison between CapsGNN-Avg-noRout and CapsGNN-noRout on NCI1, PROTEINS and

D&D indicates the effectiveness of Attention Module which improves the accuracy by 1.37%, 1.24% and 1.35% respectively. The accuracy of NCI1, PROTEINS and D&D can be improved by as much as 2.98%, 1.33% and 2.23% when Attention Module and Routing mechanism are combined in the architecture.

Overall, CapsGNN is a general framework that fuses capsule theory to GNN for more efficient graph representation learning. In this framework, we also provide multiple possible modules to improve the quality of learned graph embeddings while we do not target to find the best combination of modules for each dataset. Since each possible module plays a different role in different datasets, it would be better to adjust the architecture and hyper-parameters based on practical situation.

4.6 Conclusion

In this chapter, we have presented CapsGNN, a novel framework that fuse capsules theory into GNN for more efficient graph representation learning. Inspired by CapsNet, the concepts of capsules are introduced in this architecture to extract features in the form of vectors on the basis of nodes features extracted from GNN. As a result, one graph is represented as multiple embeddings and each embedding captures different aspects of the graph properties. The generated graph and class capsules can preserve not only the classification-related information but also other information with respect to graph properties which might be useful in the follow-up work and we leave this to be explored in the future. This is a novel, efficient and powerful data-driven method to represent high-dimensional data such as graphs. Our model has successfully achieved better or comparable performance when compared with other state-of-the-art algorithms on 6 out of 10 graph classification tasks especially on social datasets. Compared with similar scalar-based architectures, CapsGNN is more efficient in encoding features and this would be very beneficial for processing large datasets. All the related code is available at <https://github.com/XinyiZ001/CapsGNN> and more graph/class capsule analysis figures are available at <https://sites.google.com/view/capsgnn/home>.

Chapter 5

mSHINE: A Multiple-meta-paths Simultaneous Learning Framework for Heterogeneous Information Network Embedding

5.1 Introduction

This Chapter focuses on node-level representation learning, especially for the HIN which is receiving increasing attention because of its strong capability of describing objects with rich information [26]. However, the heterogeneity of HINs also raises the problem of potential semantic incompatibility which might affect the performance of network embeddings in downstream tasks. For example, Figure 3.2a illustrates a simple HIN where users are connected with their reviewed movies, some attributes (actors, directors and genres) of these movies are also listed. If we analyse the relationship between U_1 and U_3 , it is easy to find that they reviewed movies produced by the same directors. However, as for U_1 and U_2 , they reviewed movies with the same actors. So we may conclude that U_1 and U_2 are interested in the same actors while U_1 and U_3 are interested in the same directors. This reflects different types of semantic information of HINs which can also be referred as different aspects of HINs. If we focus on actors, U_1 should be closer to U_2 while it should be closer to U_3 if we focus on directors. In this case, directly representing

each user with a single lower-dimensional representation may lead to information loss and reduce the quality of network embeddings.

Since traditional homogeneous information network embedding methods [13–15, 86] fail to capture information included in HINs from different aspects, some new methods [4, 22, 28, 44, 45, 49, 71, 95, 96] that are designed for HIN embedding have been proposed to address the problems mentioned above. Many of these works make use of meta-paths [21] to distinguish different semantic information and have verified the effectiveness of meta-paths in HIN embedding. However, meta-path selection is still a challenge in current research works. Some methods [4, 22] require selecting meta-paths in advance based on experts’ experience and some other methods [28, 95] require setting the limitation of meta-paths’ length and then all possible meta-paths will be greedily examined. These methods are hard to be applied in practical applications when there is no prior knowledge from domain experts for optimal meta-path selection. Other tools [49, 96] instead of meta-paths can also be applied to represent different aspects of an HIN while the practical meaning of these aspects are sometimes unexplainable. Besides, some hyper-parameters are still required to be set to select valid aspects for practical use. Another challenge is, for each of the selected meta-path (aspect), re-learning all node representations is required and this is extremely inefficient especially when a large number of meta-paths (aspects) or meta-path (aspect) combinations need to be tested.

To address the problems mentioned above, we propose **mSHINE** (a **m**ultiple-**m**eta-paths **S**imultaneous learning framework for **H**eterogeneous **I**nformation **N**etwork **E**MBEDDING) for HIN representation learning. This framework mainly consists of two parts: *Initial meta-path selection module* and *Node representation learning module*. Firstly, to reduce the cost of meta-path exploration during the training stage, a set of criteria are proposed to select valid initial meta-paths that are used for training. Then the node representation learning module, which is inspired by the concept and basic structure of recurrent neural network (RNN) [31], is applied to capture long-term node information. At the same time, combined with meta-paths information, the representation learning module is designed to be capable of simultaneously learning the respective node representations for all selected initial meta-paths. Finally, node representations learned from different meta-paths can

be applied to the target tasks. Users can further choose the most suitable meta-path which achieves the best performance for a specific task. In addition to the learned node representations, the trained framework can also be easily applied in downstream link prediction tasks. The main contributions of this work are as follows:

- Proposed mSHINE, a novel HIN representation learning framework includes two key components: node representation learning module and initial meta-path selection module. The effectiveness of the proposed mSHINE framework is verified via extensive experiments study on 5 real-world datasets.
- The representation learning module cooperating with the proposed **shared node representations** facilitate **simultaneously learning** multiple node representations for all selected meta-paths. This improves the feasibility and learning efficiency when multiple meta-paths need to be considered.
- The representation learning module combines the concept of meta-path with RNN-inspired structure to **capture different semantic and long-term node information**, thereby solving the problem of incompatibility of HINs to a certain extend.
- The initial meta-path selection module, consisting of a set of selection criteria, is proposed to **select initial meta-paths**. This module is helpful to reduce the meta-path exploration cost.
- The well trained mSHINE **can be easily applied in downstream HIN analysis tasks** such as link prediction and node classification.

5.2 Preliminary

In this section, we provide a brief introduction to concepts and the techniques used in this work.

5.2.1 Related Concepts

Considering that the definition of HIN and network schema have been introduced in Section 3, here, we just introduce meta-path and its related concept which is a widely used technique for handling the heterogeneity of HIN.

Definition 5.1. Meta-path. A meta-path is a sequence of node and edge types that defines how two nodes in a HIN are connected and it is denoted in the form of $T_1^v \xrightarrow{T_1^e} T_2^v \xrightarrow{T_2^e} \dots T_{l-1}^v \xrightarrow{T_{l-1}^e} T_l^v$. A meta-path is also referred as symmetric meta-path if the type sequence it forms is symmetric. Meta-path has shown its effectiveness in many existing HIN analysis algorithms [21, 22, 28] and each meta-path can be also considered as an aspect or view of the HIN in these algorithms.

Definition 5.2. Meta-path-based Random Walk. Following some random walk based homogeneous information network embedding algorithms [13, 14], some HIN embedding algorithms [22] propose to generate node sequences under the direction of meta-paths which is referred as meta-path-based random walk. To be more specifically, given a HIN $\mathcal{G} = (\mathbb{V}, \mathbb{E})$ and a meta-path $M_j : T_1^v \xrightarrow{T_1^e} T_2^v \xrightarrow{T_2^e} \dots T_{l-1}^v \xrightarrow{T_{l-1}^e} T_l^v$, when generating a node sequence for training, the transition probability at the $(i + 1)$ -th step is:

$$p(n_{i+1}|n_i, M_j) = \begin{cases} \frac{1}{|\mathcal{N}^{T_{i+1}^v}(n_i)|}, & (n_i, n_{i+1}) \in \mathbb{E} \text{ and } \phi(n_{i+1}) = T_{i+1}^v \\ 0, & \text{otherwise} \end{cases} \quad (5.1)$$

where $\mathcal{N}^{T_{i+1}^v}(n_i)$ represents the neighbor nodes of n_i with the node type of T_{i+1}^v . It is noted that when the given meta-path M_j is a symmetric meta-path, meta-path-based random walk could repetitively follow the pattern provided by M_j to generate the following node sequence until it reaches a predefined node sequence length. For example, if we do random walk on the HIN which is shown in Figure 3.2a based on the symmetric meta-path $U \rightarrow M \rightarrow A \rightarrow M \rightarrow U$, the extracted node sequences can be $U_1 \rightarrow M_1 \rightarrow A_2 \rightarrow M_2 \rightarrow U_2 \rightarrow M_5 \rightarrow A_1 \rightarrow M_1 \dots$.

5.2.2 Problem Statement

Given a HIN $\mathcal{G} = (\mathbb{V}, \mathbb{E})$, mSHINE intends to find a set of initial meta-paths $\mathbb{M} = \{M_1, M_2, \dots, M_n\}$ and a function $f : \mathbb{V} \rightarrow \mathbb{R}_{M_i}^d$ to map each node to its feature

representation associated with the meta-path M_i . The important information of $\mathcal{G}$ should be well preserved in the learned node representations.

5.3 Related Work

5.3.1 Negative Sampling

Negative sampling is firstly proposed in Word2vec [37] which is designed to address the inefficiency of softmax when it is applied to process a very high-dimensional data. Specifically, to preserve the connectivity information between node n_i and its neighbor nodes $n_c \in \mathcal{N}(n_i)$, the corresponding objective function should be:

$$\frac{1}{|\mathcal{N}(n_i)|} \sum_{n_k \in \mathcal{N}(n_i)} \log p(n_k | n_i) \quad (5.2)$$

where

$$p(n_k | n_i) = \frac{\exp(\vec{v}_{n_k}^T \vec{v}_{n_i})}{\sum_{n_t \in \mathbb{V}} \exp(\vec{v}_{n_t}^T \vec{v}_{n_i})} \quad (5.3)$$

where $\vec{v}_{n_i}$ and $\vec{v}_{n_k}$ indicates the node representations of n_i and n_k respectively. Given n_i , directly optimizing parameters based on the Equation (5.2) which is calculated from softmax, means the dot product of all the node representations and $\vec{v}_{n_i}$ need to be calculated. This procedure is extremely inefficient. So negative sampling is proposed to replace $\log p(n_k | n_i)$ with:

$$\log \sigma(\vec{v}_{n_k}^T \vec{v}_{n_i}) + \sum_{t=1}^T \mathbb{E}_{n_t \sim P_n(n)} [\log \sigma(-\vec{v}_{n_t}^T \vec{v}_{n_i})] \quad (5.4)$$

where T indicates the number of negative samples we selected each time for training. In this case, only a small subset of nodes (T) that do not belong to $\mathcal{N}(n_i)$ are randomly selected at each iteration to updates their node representations. After converge, the connectivity of $\mathcal{G}$ can be well reflected through the learned node representations.

mSHINE is also designed to be trained by preserving the node connectivity, therefore, negative sampling is applied in mSHINE for improving the training efficiency when calculating the probability that a node pair is connected.

FIGURE 5.1: The structure of a simple RNN unit.

5.3.2 Recurrent Neural Network

Recurrent neural network, known as RNN, is designed to capture the temporal dynamic information with any length. It is widely used to process sequential data such as speech, text, node sequence, and so on. Although there are a lot of variants of RNN such as LSTM, GRU, Bi-LSTM, and so on, this work chooses to use the simplest version: Elman Network [31] to show the feasibility of the main idea. The structure of the simple RNN can be found in Figure 5.1 and the procedure can be formally written as:

$$\vec{h}_t = \sigma(W_x \vec{x}_t + W_h \vec{h}_{t-1} + \vec{b}_h) \quad (5.5)$$

$$\vec{y}_t = \sigma(W_y \vec{h}_t + \vec{b}_y) \quad (5.6)$$

In mSHINE, we design an RNN-inspired structure to process the topological information around each node by converting it to the sequential information. Specifically, for each node, we recurrently apply the unit to capture the information from its neighbors, when the number of recurrent times is sufficient, the topological information around each node can be well captured.

5.4 Methodology

In this section, we present mSHINE and show how it applies RNN inspired structure to capture long-term node information and simultaneously learn node representations from multiple meta-paths. Besides, the way we select valid meta-paths is also introduced in this section.

The overall structure of mSHINE is shown in Figure 5.2 which mainly consists of two modules and we will introduce these modules in detail respectively: 1) **Node**

FIGURE 5.2: The schematic illustration of mSHINE. Meta-path selection criteria are applied to select valid meta-paths first and these valid meta-paths are used to direct sampling three-element node sequences. For each three-element node sequence $n_{i-1} \xrightarrow{T_a^e} n_i \xrightarrow{T_b^e} n_{i+1}$, the general basic node representation $\vec{x}_{n_i}$ for n_i and general state node representations $\vec{h}_{n_{i-1}}, \vec{h}_{n_i}$ for n_{i-1} and n_i are generated through looking up the basic/state node representation tables. Then meta-path-related functions are applied to process the general basic/state node representations to generate $\vec{x}_{n_i}^{M_j}, \vec{h}_{n_{i-1}}^{M_j}$ and $\vec{h}_{n_i}^{M_j}$ respectively. The processed node representations ($\vec{x}_{n_i}^{M_j}$ and $\vec{h}_{n_{i-1}}^{M_j}$) are then fed into RNN-inspired unit together with the edge representation $\vec{e}_{T_{ab}^e}$ to generate a new state node representation $\vec{h}_{n_i, T_{ab}^e}^{M_j}$ for n_i as well as the prediction of the next node n_{i+1} . All the trainable parameters in the proposed structure are trained based on $\mathcal{L}_{state}(n_i)$ and $\mathcal{O}(n_{i+1})$. Finally, the basic node representations as well as meta-path-related functions can be used to generate meta-path-related node representations and the trained node representation learning module can be applied in downstream link prediction tasks.

Representation Learning module and 2) **Initial Meta-path Selection** module.

5.4.1 Node Representation Learning

As introduced in Section 5.1, traditional network embedding methods such as DeepWalk [13] and Node2vec [14] fail to capture different semantic information of HINs

since they simply use random walk to generate node sequences for node representation learning. Hence, we choose to extract meaningful node sequences under the direction of meta-paths for node representation learning to distinguish different semantic information. This is similar to Metapath2vec [22] where a HIN is converted to a stream of node sequences through meta-path-based random walk. By considering these extracted node sequences as sentences and nodes as words, language modeling methods can be applied to explore the property of the original HIN. Different from Metapath2vec [22] where Word2vec [37] is chosen as the language modeling method, the node representation learning module in mSHINE is developed by adapting the working principle and basic structure of RNN which is designed for processing sequential data. The detailed structure of this representation learning module will be introduced from two levels: 1) **Structure Information Modelling** which is designed to capture connection information from extracted node sequences. and 2) **Semantic Information Modelling** which is developed on the basis of structure information modeling to include semantic information.

5.4.1.1 Structure Information Modelling

At the structure information modeling level, the edge and node types are ignored and we only focus on the connection information at this level.

In the original RNN structure, at each timestamp t , an RNN unit takes the current *input* $\vec{x}_t$ and the *hidden state* $\vec{h}_{t-1}$ to predict the current *output* $\vec{y}_t$ and generate a new *hidden state* $\vec{h}_t$. In this work, when we process an extracted node sequence (the details about how to select initial meta-path and how to extract meta-path-based node sequences will be explained later) with RNN structure, the node sequence should be decomposed into multiple triple sequences and each triple sequence corresponds a timestamp. For example, supposing the node sequence is $n_1 \rightarrow n_2 \cdots n_{l-1} \rightarrow n_l$ and it should be decomposed into a triple sequence set $\{n_1 \rightarrow n_2 \rightarrow n_3, n_2 \rightarrow n_3 \rightarrow n_4, \cdots, n_{l-2} \rightarrow n_{l-1} \rightarrow n_l\}$, for each triple sequence in this set, the processing procedure is shown in Figure 5.1 and formally, it is written as:

$$\vec{h}_{n_i} = \sigma(W_{xh}\vec{x}_{n_i} + W_{hh}\vec{h}_{n_{i-1}}) \quad (5.7)$$

$$\vec{y}_{n_i} = W_{hy}\vec{h}_{n_i} \quad (5.8)$$

$$p(n_{i+1}|n_i \cdots n_1) = \frac{\exp(\vec{y}_{n_i, n_{i+1}})}{\sum_{v' \in \mathbb{V}} \exp(\vec{y}_{n_i, v'})} \quad (5.9)$$

where there are some minor modifications compared with Equation (5.5) and Equation (5.6), to adapt to the character of node representation learning in HIN. Specifically, the input of this RNN unit should be the basic representation of n_i ($\vec{x}_{n_i}$) and the state representation of n_{i-1} ($\vec{h}_{n_{i-1}}$). The output includes a new state representation $\vec{h}_{n_i} \in \mathbb{R}^{d \times 1}$ for n_i as well as the probability of the existing of n_{i+1} which is denoted as $p(n_{i+1}|n_i \cdots n_1)$ and it is calculated from $\vec{y}_{n_i} \in \mathbb{R}^{N \times 1}$. $W_{xh} \in \mathbb{R}^{d \times d}$ and $W_{hh} \in \mathbb{R}^{d \times d}$ denote the trainable transform matrices which are used to do transformation between the basic representation domain and the state representation domain. d represents the dimension of node representations. $\vec{y}_{n_i, v'}$ denotes the possibility of the existing of node v' based on the prediction of node sequence $n_i \cdots n_1$. $W_{hy} \in \mathbb{R}^{N \times d}$ denotes the output representation matrix which contains all the target node representations and N here represents the number of nodes in this HIN. $\sigma(\cdot)$ is the activation function.

So far, by optimizing the probability of the existing of n_{i+1} , the structural information of the extracted node sequences can be captured by the basic RNN-inspired structure and preserved in the learned basic and state node representations.

5.4.1.2 Semantic Information Modelling

In HINs, besides structural information, semantic information which is provided by the attributes of different edges and nodes also plays an important role in describing object properties. Based on the basic representation learning module introduced in Section 5.4.1.1, we will include more semantic information at this level to enhance the learned node representation quality. This level mainly consists of two parts: 1) **Meta-path-based Semantic Information** and 2) **Edge-based Semantic Information**.

Meta-path-based Semantic Information. As introduced in Table 5.2, node representations that are learned based on different meta-paths can reflect different aspects of this node and separately embedding these aspects is helpful to describe different properties of this node. So, when processing a node sequence that is extracted based on the meta-path M_j , the corresponding meta-path information should be embedded into the representation of each node in this sequence. This

is achieved through the meta-path-related functions and the procedure can be formally written as:

$$\vec{x}_{n_i}^{M_j} = f_x^{M_j}(\vec{x}_{n_i}) \quad (5.10)$$

$$\vec{h}_{n_i}^{M_j} = f_h^{M_j}(\vec{h}_{n_i}) \quad (5.11)$$

Here, $\vec{x}_{n_i}$ and $\vec{h}_{n_i}$ denote the general basic node representation and general state node representation of n_i respectively. The word *general* here indicates that these representations contain node information from all meta-paths and they are independent from any specific meta-paths. Then meta-path-related functions (i.e., $f_x^{M_j}(\cdot)$ and $f_h^{M_j}(\cdot)$) are used to decode information that is related to the meta-path M_j from $\vec{x}_{n_i}$ and $\vec{h}_{n_i}$ respectively. As a result, $\vec{x}_{n_i}^{M_j}$ and $\vec{h}_{n_i}^{M_j}$ can be used to reflect the meta-path-related properties of n_i and these meta-path-related node representations can be easily used in downstream HIN analysis tasks.

Edge-based Semantic Information. As introduced in Section 5.4.1.1, a node sequence which is extracted based on the meta-path M_j can be further decomposed into multiple triple sequences ($n_{i-1} \xrightarrow{T_a^e} n_i \xrightarrow{T_b^e} n_{i+1}$). The triple node sequence type T_{ab}^e (a.k.a edge-based information) is also a part of the crucial information which needs to be embedded into the representation of each node that appears in this triple sequence. To include both edge-based and meta-path-based information into the node representations, the procedure which is originally written as Equation (5.7)-Equation (5.9) can be further modified as:

$$\vec{h}_{n_i, T_{ab}^e}^{M_j} = \sigma(W_{xh}\vec{x}_{n_i}^{M_j} + W_{hh}\vec{h}_{n_{i-1}}^{M_j} + W_{rh}\vec{e}_{T_{ab}^e}) \quad (5.12)$$

$$\vec{y}_{n_i, T_{ab}^e}^{M_j} = W_{hy}^{M_j} \vec{h}_{n_i, T_{ab}^e}^{M_j} \quad (5.13)$$

where

$$W_{hy}^{M_j} = f_y^{M_j}(W_{hy}) \quad (5.14)$$

and

$$p(n_{i+1}|n_i \cdots n_1, M_j, T_{ab}^e) = \frac{\exp(\vec{y}_{n_i, T_{ab}^e, n_{i+1}}^{M_j})}{\sum_{v' \in \mathbb{V}, \phi(v') \in \phi(n_{i+1})} \exp(\vec{y}_{n_i, T_{ab}^e, v'}^{M_j})} \quad (5.15)$$

where $\vec{e}_{T_{ab}^e} \in \mathbb{R}^{d \times 1}$ is used to decode the edge-based semantic information from node representations and it is decided by the triple node sequence type T_{ab}^e . The

role of $\vec{e}_{T_{ab}^e}$ can be considered as the same as the context vector $\vec{c}$ in RNN Encoder–Decoder work [97]. Since the triple node sequence type T_{ab}^e is uniquely determined by the node sequence $n_i \cdots n_1$, $p(n_{i+1}|n_i \cdots n_1, M_j, T_{ab}^e)$ can be directly written as $p(n_{i+1}|n_i \cdots n_1, M_j)$ in the following discussion. Similar to Equation (5.9), $\vec{y}_{n_i, T_{ab}^e, v'}^{M_j}$ denotes the possibility of the existing of node v' predicted from $\vec{h}_{n_i, T_{ab}^e}^{M_j}$. Besides, it is worth noting that W_{hy} is row-wisely processed with its corresponding meta-path-related function $f_y^{M_j}(\cdot)$. This is because each row of W_{hy} is the target representation of a node and it is more reasonable to apply meta-path-related function to these target node representations too.

By including both meta-path-based and edge-based semantic information, the structure introduced above is capable of capturing both connection information as well as heterogeneous semantic information of extracted node sequences.

5.4.1.3 Optimization Objective

Multiple optimization objectives are designed to learn the trainable parameters as well as node representations in the proposed model.

The first optimization objective is the prediction objective. In specific, given a node sequence $s^{i+1} = (n_{i+1}, n_i, \cdots n_1)$ which is extracted based on meta-path M_j , the prediction optimization objective is to maximize the log likelihood of n_{i+1} given its previous node sequence $s^i = (n_i, n_{i-1}, \cdots n_1)$:

$$\mathcal{O}_{pre}(n_{i+1}, M_j) = \sum_{s \in \mathbb{D}_{v' \rightarrow n_i}^{M_j}, v' \in \mathbb{V}} \log p(n_{i+1}|s^i, M_j) \quad (5.16)$$

where $D_{u \rightarrow v}^{M_j}$ denotes node sequence set which are extracted based on meta-path type M_j and these sequences should start from node u and end at node v . Here, the probability $p(n_{i+1}|s^i, M_j)$ is defined as Equation (5.15). As introduced in Section 5.2, given a meta-path M_j , the extracted meta-path-based node sequences can be any length (if M_j is a symmetric meta-path), therefore, the length of s which is extracted based on M_j does not have to be i , so $p(n_{i+1}|s^i, M_j)$ is directly written as $p(n_{i+1}|s, M_j)$ in the following part. As directly optimizing Equation (5.16) is computationally expensive, negative sampling [37] is used here to improve training

efficiency. Hence, Equation (5.15) can be approximated and written as:

$$\log p(n_{i+1}|s, M_j) \approx \log \sigma(W_{hy, n_{i+1}}^{M_j} \vec{h}_{n_i, T_{ab}^e}^{M_j}) + \frac{1}{K} \sum_{\substack{t=1, \\ \phi(n_t)=\phi(n_{i+1})}}^K \log \sigma(-W_{hy, n_t}^{M_j} \vec{h}_{n_i, T_{ab}^e}^{M_j}) \quad (5.17)$$

where K denotes the number of negative samples and $W_{hy, n_t}^{M_j}$ denotes the target node representation of negative node n_t under meta-path M_j .

As introduced above, prior to learning node representations with the proposed structure, a large number of node sequences ($\mathbb{D}$) should be prepared in advance by conducting meta-path-based random walk. Then for each node sequence s , an optimization algorithm is applied to maximize the objective value which is calculated based on Equation (5.17). However, as the node sequence length grows, training parameters based on the objective function becomes inefficient. So we proposed to relax the procedure of calculating state representation of each node so that processing a long node sequence to generate state representation for each node is not required anymore. To be specific, given a triple sequence $n_{i-1} \xrightarrow{T_a^e} n_i \xrightarrow{T_b^e} n_{i+1}$ and its corresponding meta-path M_j , it is not required to calculate $\vec{h}_{n_{i-1}}^{M_j}$ based on its previous triple node sequences, instead, $\vec{h}_{n_{i-1}}^{M_j}$ can be directly obtained by looking up the stored general state representations which is then processed by a meta-path-related function $f_h^{M_j}(\cdot)$. In this case, the structure used in the node representation learning module becomes quite different from the original RNN and we call the structure as RNN-inspired structure simply because it has state node representations. In HIN representation learning, when predicting n_{i+1} based on n_{i-1} and n_i , the state representation of n_{i-1} should actually provide its spatial environment information instead of just the sequential information generated based on only one single node sequence. So using stored general state representations which contain rich information captured from various node sequences is more desirable in this case. As a result, in addition to training the stored general state representations with prediction objective value, we also set the second optimization objective to embed newly generated information into stored state representations: minimizing the distance between the meta-path-related state representation $\vec{h}_{n_i}^{M_j}$ which is generated through looking up stored representations and the newly generated meta-path-related state representation $\vec{h}_{n_i, T_{ab}^e}^{M_j}$ which is calculated through Eq.

FIGURE 5.3: An example of decomposing a meta-path into the corresponding triple node sequence type set.

(5.12). This distance can be calculated through:

$$\mathcal{L}_{state}(n_i, M_j) = \|\vec{h}_{n_i}^{M_j} - \vec{h}_{n_i, T_{ab}^e}^{M_j}\|_{L_2} \quad (5.18)$$

Storing general state representations of nodes makes sampling a huge number of long node sequences for training unnecessary, so when preparing training data, we only need to sample **triple node sequences** around each node and find the corresponding meta-path type for each of the triple node sequences.

Overall, the optimization objective of the proposed method can be formally written as:

$$\mathcal{O}_{all} = \sum_{M_j \in \mathbb{M}} [\sum_{n_i \in \mathbb{V}} \mathcal{O}_{pre}(n_i, M_j) - \sum_{n_i \in \mathbb{V}} \mathcal{L}_{state}(n_i, M_j)] \quad (5.19)$$

5.4.2 Initial Meta-path Selection

Initial meta-paths need to be selected in advance for training in this work. It is worth noting that initial meta-paths can be selected based on prior knowledge from domain experts if it is available. However, in most of the case, prior knowledge is unavailable in HIN representation learning. So in this section, the selection of valid initial meta-paths without prior knowledge is introduced.

As discussed in Section 5.4.1.3, the training of node representations for a meta-path M_j is actually associated with a set of triple node sequence types $\mathbb{C}_{M_j}$, and the learned node representations will be the same for meta-path M_j and meta-path M_k if their decomposed triple node sequence type set $\mathbb{C}_{M_j}$ and $\mathbb{C}_{M_k}$ are the same. So two meta-paths are considered as the same if their decomposed triple node sequence type sets are the same and we only keep the shorter meta-path during

initial meta-path selection. The procedure of analysing the triple node sequence type set $\mathbb{C}_{M_j}$ from a meta-path M_j is illustrated in Figure 5.3. Specifically, based on a meta-path $M_j = U \rightarrow M \rightarrow A \rightarrow M \rightarrow U$, a long node sequence can be extracted: $u_1 \rightarrow m_1 \rightarrow a_1 \rightarrow m_1 \rightarrow u_2 \rightarrow m_2 \rightarrow a_3 \rightarrow m_3 \rightarrow u_3 \cdots$. Overall, there are 4 three-element node sequence types that appeared in this node sequence which are $U \rightarrow M \rightarrow A$, $M \rightarrow A \rightarrow M$, $A \rightarrow M \rightarrow U$ and $M \rightarrow U \rightarrow M$. So, the meta-path M_j can be decomposed into a triple node sequence type set $\mathbb{C}_{M_j} = \{ U \rightarrow M \rightarrow A, M \rightarrow A \rightarrow M, A \rightarrow M \rightarrow U, M \rightarrow U \rightarrow M \}$. Noted that according to previous work [4, 21], symmetric meta-paths are more suitable for describing the similarity between two nodes with the same node types. So, we only select symmetric meta-paths as valid initial meta-paths in this paper to illustrate the idea although the proposed framework also works for asymmetric meta-paths.

Based on the discussion above, the criteria for selecting initial meta-paths for training are summarized as below:

- 1) Only symmetric meta-paths are selected as valid meta-paths.
- 2) Two meta-paths are considered as the same if their decomposed triple node sequence type sets are the same and the shorter one is selected as a valid meta-path.
- 3) Any of the selected meta-paths should not be included by any other valid meta-paths.

In a HIN, the number of possible triple node sequence types is limited. So, the number of meta-paths that meet the proposed criteria is also limited and we will select all meta-paths that satisfy the criteria as initial meta-paths in mSHINE. The node representation module introduced in Section 5.4.1 is then applied to learn node representations for all initial meta-paths simultaneously. After training, node representations corresponding to different meta-paths can be easily obtained by processing general node representations with different meta-path-related functions, then we can select the most suitable meta-paths and apply the respective node representations based on specific tasks. The overall framework of mSHINE which consists of initial meta-paths selection criteria and node representation learning module is specified in Algorithm 2.

Algorithm 2 mSHINE

Input: $\mathcal{G}$: Information network.
Output: $\mathbf{X}$: Basic node representations; f_x^M : Meta-path-related functions.

```

1: procedure LEARNINGMODULE( $\mathcal{G}$ )
2:   if Prior knowledge is available then
3:     Construct valid initial meta-path set  $\mathbb{M}$  based on prior knowledge;
4:   else
5:     Construct valid initial meta-path set  $\mathbb{M}$  based on the proposed criteria
    in Section 5.4.2;
6:   end if
7:   Initialize  $\mathbf{X}$ ,  $\mathbf{H}$  and other trainable parameters;
8:   while not converged do
9:     for  $M_j \in \mathbb{M}$  do ▷ Select a meta-path type.
10:      Obtain  $\mathbb{C}_{M_j}$  based on  $M_j$ ;
11:      for  $T_{ab}^e \in \mathbb{C}_{M_j}$  do ▷ Select a triple node sequence type.
12:        Sample a triple sequence for training  $s \in \mathbb{D}^{M_j}$  ( $n_{i-1} \xrightarrow{T_a^e} n_i \xrightarrow{T_b^e} n_{i+1}$ ).
13:        Sample negative samples  $\tilde{n}_{i+1}$ ;
14:        Start ▷ Meta-path-related processing.
15:           $\vec{h}_{n_{i-1}}^{M_j} \leftarrow f_h^{M_j}(\vec{h}_{n_{i-1}})$ ;
16:           $\vec{x}_{n_i}^{M_j} \leftarrow f_x^{M_j}(\vec{x}_{n_i})$ ;
17:           $\vec{h}_{n_i}^{M_j} \leftarrow f_h^{M_j}(\vec{h}_{n_i})$ ;
18:           $W_{xy, n_{i+1}}^{M_j} \leftarrow f_y^{M_j}(W_{xy, n_{i+1}})$ ;
19:           $W_{xy, \tilde{n}_{i+1}}^{M_j} \leftarrow f_y^{M_j}(W_{xy, \tilde{n}_{i+1}})$ ;
20:        End
21:        Calculate objective value with Eq.(5.19);
22:        Updating trainable parameters by maximizing the objective
        value;
23:      end for
24:    end for
25:  end while
26:  return  $\mathbf{X}, f_x^M$ ;
27: end procedure

```

5.5 Experiments

5.5.1 Datasets

We use five publicly available real-world datasets: Douban Movie 1, DBLP, Cora, IMDB and Yelp to form HIN in the experiments¹. The detailed description of those

¹All the node features are ignored in this chapter's experiments

datasets can be found in Section 3.2 and all edges in these datasets are considered as undirected edges.

5.5.2 Baseline Methods

To provide comprehensive evaluation, we tested a number of state-of-the-art methods such as GraphSage [25], HeGAN [71] as well as polysemous embedding [96] and we choose to report the most relevant 5 algorithms which are DeepWalk, LINE, Metapath2vec, AspEm and HIN2vec. We evaluate the effectiveness of the proposed mSHINE against these 5 baseline representation learning methods on two HIN analysis tasks: nodes classification and link prediction. Here, DeepWalk and LINE are originally designed to process homogeneous information networks. In our experiments, these two methods are applied by ignoring the node and edge types in HIN.

DeepWalk [13] is the first method that applied NLP algorithms(Word2vec [37]) in information network representation learning tasks. Specifically, a number of paths are extracted started from each node through random walk, then each path is considered as a sentence and each node is considered as a word. Word2vec then is applied to learning the representation of each node.

LINE [15] is proposed to preserve both *local* and *global* network structure information through a carefully designed objective function. Different from DeepWalk, the training procedure of LINE is based on edge-sampling and this is more efficient than random-walk-based training which is used in DeepWalk and random-walk-related algorithms.

Metapath2vec [22] is proposed based on DeepWalk and mainly designed for HIN representation learning. There are two main differences compared with DeepWalk: 1) random walk is conducted under the direction of Metapath in Metapath2vec; 2) Metapath2vec applies heterogeneous softmax(negative sampling) at the prediction layer.

AspEm [49] proposed a new coefficient to measure the incompatibility of different types of edges in an HIN. When the incompatibility of a group of edge types is smaller than a pre-defined threshold, this group of edge types will be considered as an aspect of the HIN and then node representations will be learned within different

aspects separately. AspEm does not specify the representation learning algorithm in the original paper, we choose to use the PTE [45] in our experiments which is also suggested by AspEm.

HIN2vec [28] proposed to limit the length of possible meta-paths and then learn node and meta-path representations. In this algorithm, a number of paths are extracted through random walk first and then node pairs within a predefined window size as well as the meta-path type that connects these two nodes are taken as a training sample. The learning structure of HIN2vec is designed to predict the existing of a specific type of connection between two nodes.

The selected 5 baselines have a comprehensive coverage of existing HIN representation learning methods. DeepWalk and LINE are the earliest and classic homogeneous information network representation learning methods. Metapath2vec, AspEm and HIN2vec are the typical HIN representation learning methods where Metapath2vec and HIN2vec make use of meta-paths to distinguish different semantic information of HIN while AspEm extracts different aspects by analyzing the structural information of HIN to address the problem of incompatibility.

5.5.3 Implementation Details

The implementation details of mSHINE in the experiments are shown as below which is applied on all the datasets:

- a) The number initial meta-paths we selected based on the criteria proposed in Section 5.4.2 for all experimental datasets are summarized in Table 5.1.
- b) As for meta-path-related functions $f_x^{M_j}$, $f_h^{M_j}$ and $f_y^{M_j}$, we choose to use Hadamard product in this work:

$$\vec{x}_{n_i}^{M_j} = f_x^{M_j}(\vec{x}_{n_i}) = \vec{x}_{n_i} \circ \vec{v}_x^{M_j} \quad (5.20)$$

$$\vec{h}_{n_i}^{M_j} = f_h^{M_j}(\vec{h}_{n_i}) = \vec{h}_{n_i} \circ \vec{v}_h^{M_j} \quad (5.21)$$

$$W_{hy,n_i}^{M_j} = f_y^{M_j}(W_{hy,n_i}) = W_{hy,n_i} \circ \vec{v}_y^{M_j} \quad (5.22)$$

where $\vec{v}_h^{M_j}, \vec{v}_x^{M_j}, \vec{v}_y^{M_j} \in \mathbb{R}^{d \times 1}$ are trainable vectors. Other meta-path-related functions can also be used while we just take Hadamard product as an example.

TABLE 5.1: The Number of Initial Meta-paths for Each Dataset

Dataset	Douban Movie 1	DBLP	Cora	IMDB	Yelp
Num. of Meta-paths	15	6	6	10	10

- c) Batch size is set as $B = 30$ for each iteration and each training sample should provide a triple node sequence as well as the corresponding meta-path type.
- d) The dimension of node representations(d) and negative sampling rate(K) are set as 128 and 5 respectively. Other algorithms also follow this setting.
- e) The number of epochs is set as $E = 1000$ for all datasets.
- f) Stochastic gradient descent (SGD) [98] is used to train the parameters in mSHINE.
- g) When training finished, the learned basic meta-path-related node representations are used in experimental studies to show the performance of mSHINE².
- h) Other hyper-parameters are set as the same as other baseline algorithms which will be introduced individually in each experiment section.
- i) All the experiments are conducted in Nvidia Tesla P100 Cluster.

5.5.4 Node Classification

In this section, the effectiveness of mSHINE is evaluated through node classification. Related experimental setup as well as the procedure of constructing classification datasets will be introduced first and then the experimental results will be discussed.

5.5.4.1 Experimental setup

Data preparation. To evaluate the performance of algorithms through node classification task, HIN datasets which are used for node classification should provide node labels, so, Cora, Yelp and Douban Movie 1 are used as the experimental datasets in this experiment. As introduced in Section 5.5.1, papers in Cora can

²Before training, we use normal distribution to initialize the basic node representations where $mean = 0$ and $stddev = 0.1$. The state and output node representations are initialized as 0.

be categorized by 10 different research areas and these areas are used as paper labels for paper node classification (the property of DBLP is similar to Cora and we choose to use Cora because of its larger dataset size). In yelp, we select 10 main cuisines in restaurants' categories as labels and there are 17,782 restaurant business can be labeled with one of the 10 cuisine types. Besides, we use Douban Movie 1 dataset to evaluate the quality of the learned node representations in multi-label classification task where movie types will be used as movie labels. Thus, the labeled node classification datasets are constructed.

Hyper-parameter setting. For random-walk-based baseline algorithms, for fairly comparing their performance, the number of node sequences that are extracted from each node is set as the same which is $w_n = 80$ and the sequence length is set as $w_l = 40$ which are the same as DeepWalk paper. As for the window size, we use $w_s = 4$ which follows the experimental setting of HIN2vec. Regarding other baseline algorithms, we follow the default setup as what is reported in the original papers. In AspEm, we follow the steps they proposed to select aspects and separately learn node representations for each aspect. At last, we report the best results after comparing all the selected aspects. As for Metapath2vec which requires us manually selecting meta-paths, we only test a limited number of meta-paths for each dataset since it is not practical to test all possible meta-paths. The way we set meta-paths for Metapath2vec follows two steps: 1) decide key node types. For example, the key node type in Cora should be *Paper* since the node classification is conducted based on paper nodes. 2) list all symmetric meta-paths which start from the key node type and end when the key node type appears at the first time. For example, the tested meta-paths of Cora are $P \rightarrow A \rightarrow P$, $P \rightarrow T \rightarrow P$ and $P \rightarrow P \rightarrow P$. The same as AspEm, when we report the results of Metapath2vec and mSHINE, we report the best results after comparing all tested meta-paths in these two methods respectively. Regarding HIN2vec, as introduced in Section 5.5.2, all meta-paths whose length are shorter than the window size $w_s = 4$ will be learned. In this case, the number of selected meta-paths(aspects) in Cora, Douban Movie 1 and Yelp is 80, 176 and 109 respectively and it is impossible to separately do classification experiments for each of the meta-paths(aspects). In this experiment, we choose to follow the original setting of HIN2vec where the general node representations are used to conduct node classification task.

Overall, we use the learned node representations from different algorithms as node

TABLE 5.2: Performance of Node Classification

Dataset	Metric	Training	DeepWalk	LINE	Metapath2vec	AspEm	HIN2vec	mSHINE
Selected Aspect					PPP	CONCAT		APPA
Cora	f1-macro	20%	0.7267	0.7304*	0.7227	0.6359	0.7172	0.7418
		40%	0.7577*	0.7573	0.7511	0.6916	0.7451	0.7662
		60%	0.7759*	0.7671	0.7645	0.7154	0.7557	0.7798
		80%	0.7811*	0.7793	0.7712	0.7315	0.7621	0.7903
	f1-micro	20%	0.7899	0.7928*	0.7875	0.7304	0.7826	0.8023
		40%	0.8118*	0.8109	0.8081	0.7620	0.8012	0.8202
		60%	0.8249*	0.8184	0.8185	0.7811	0.8093	0.8304
		80%	0.8318*	0.8277	0.8235	0.7908	0.8148	0.8378
Selected Aspect					BCaB	CONCAT		CaBUBCa
YELP	f1-macro	20%	0.4867	0.2196	0.5100*	0.4763	0.4112	0.5563
		40%	0.5231	0.2491	0.5247	0.5295*	0.4633	0.5840
		60%	0.5503*	0.2575	0.5275	0.5465	0.4876	0.5975
		80%	0.5566	0.2632	0.5360	0.5601*	0.5035	0.6123
	f1-micro	20%	0.6215	0.4269	0.6736*	0.6543	0.6017	0.6933
		40%	0.6428	0.4484	0.6790	0.6850*	0.6409	0.7066
		60%	0.6604	0.4651	0.6830	0.6963*	0.6539	0.7142
		80%	0.6646	0.4622	0.6834	0.7058*	0.6682	0.7233
Selected Aspect					MUM	CONCAT		MUGUM
Douban Movie 1	f1-macro	20%	0.0789	0.0825*	0.0612	0.0811	0.0721	0.1016
		40%	0.1070*	0.0926	0.0649	0.1002	0.0831	0.1239
		60%	0.1186*	0.1046	0.0734	0.1098	0.0900	0.1352
		80%	0.1231*	0.1111	0.0752	0.1175	0.0971	0.1448
	f1-micro	20%	0.4195	0.4575*	0.3944	0.4432	0.4397	0.4922
		40%	0.5048*	0.4645	0.4007	0.4830	0.4651	0.5157
		60%	0.5174*	0.4820	0.4298	0.4931	0.4778	0.5249
		80%	0.5196*	0.4872	0.4361	0.5013	0.4856	0.5349

features and then do node classification with off-the-shelf classifier(support vector machine(SVM) is used in our experiment). The classification performance is evaluated using *f1-macro* and *f1-micro* score respectively [99]. For all datasets, following the experimental setting of [4, 73], we set four training ratios as in {20%, 40%, 60%, 80%}. For each ratio, we randomly generate ten evaluation sets where the learned node representations are divided into training data and test data based on the training ratio and the average of these ten evaluation results on the test data will be reported as the final results.

5.5.4.2 Experimental results and discussion

The node classification results are summarized in Table 5.2. The best performance of the existing algorithms are marked with * and the best results for each dataset are highlighted in bold.

The proposed mSHINE outperforms all the state-of-the-art algorithms in node classification task which illustrates the effectiveness of the proposed method from two

angles: the capability of selecting suitable meta-paths(aspects) and the capability of capturing long-term information.

Meta-path(aspect) Selection. It is easy to find from Table 5.2 that when the test data is Yelp, HIN-based algorithms namely Metapath2vec, AspEm, HIN2vec as well as mSHINE are generally better than homogeneous-information-network-based algorithms such as DeepWalk and LINE which indicates the necessary of meta-paths(aspects) selection in Yelp node classification. So we focus on the node classification performance on Yelp to measure the capability of meta-paths(aspects) selection of an algorithm. Table 5.2 shows that a worse performance of HIN2vec compared with DeepWalk, which is expected from our understanding of HIN2vec. This algorithm simultaneously learns node and meta-path representations and mainly makes use of meta-path representations to distinguish different types of links or paths. However, from the node representation angle, all semantic information is still mixed within a single node embedding which might still introduce noise. Regarding Metapath2vec and AspEm, they are capable of embedding different semantic information into different node representations. However, for each meta-path(aspect), Metapath2vec and AspEm require re-training the entire model which is extremely inefficient. Therefore, an effective meta-path(aspect) selection strategy is necessary for these two algorithms to reduce training cost while it is still unsolved so far or the proposed strategy is sub-optimal in their work. We test more meta-paths(aspects) or meta-path(aspect) combination of Metapath2vec and AspEm to show the impact of meta-path(aspect) selection strategy and the experimental results are shown in Table 5.3. Cora and Douban Movie 1 also show the similar phenomenon as Yelp while we only display the results of Yelp here since it is the most obvious one. When Metapath2vec uses the combination of meta-paths $B \rightarrow U \rightarrow B$, $B \rightarrow C_a \rightarrow B$ and $B \rightarrow C_i \rightarrow B$, AspEm uses the combination of aspects $B - U, U - U$ and $B - C_a$, these two algorithms reach comparable performance to mSHINE where only one meta-path($C_a \rightarrow B \rightarrow U \rightarrow B \rightarrow C_a$) is used to learn node representations. However, these effective meta-paths(aspects) combination shown in Table 5.3 cannot be easily detected through their original method while mSHINE is able to address this problem in a feasible way. mSHINE learns multiple node representations from multiple meta-paths simultaneously, and then users can select the most suitable meta-path and node representations based on specific classification tasks, without extra efforts after training once.

TABLE 5.3: Performance of Node Classification with Yelp with various Meta-paths(Aspects)

Metric	Training	Metapath2vec			AspEm		
Selected Aspect		BUB	BCiB	BUB+ BCiB+BCaB	BCa+BCi	BU+UU	BU+ UU+BCa
f1-macro	20%	0.2765	0.1096	0.5575	0.5378	0.1995	0.5208
	40%	0.3110	0.1060	0.5904	0.5536	0.2189	0.5593
	60%	0.3385	0.1022	0.6085	0.5616	0.2360	0.5837
	80%	0.3440	0.1071	0.6183	0.5628	0.2478	0.5970
f1-micro	20%	0.4095	0.3070	0.7035	0.6847	0.3699	0.6801
	40%	0.4309	0.3084	0.7215	0.6961	0.3801	0.7020
	60%	0.4442	0.3089	0.7305	0.6992	0.3872	0.7131
	80%	0.4485	0.3105	0.7352	0.6997	0.3989	0.7182

Long-term information. From data Cora, it is easy to find that the performance of long-term-information-based algorithms such as DeepWalk, HIN2vec, Metapath2vec as well as mSHINE are generally better than short-term-information-based algorithms(AspEm) which indicates the importance of long-term information in Cora node classification. So the better performance of mSHINE in Cora justifies its effectiveness in capturing long-term information. In DeepWalk, HIN2vec and Metapath2vec, a larger window size indicates a longer long-term information the algorithms are capturing. The influence of window size to the classification performance on Cora is shown in Table 5.4. We highlight the best window size for each algorithm where we only compare the results of window size ranges from $w_s = 1$ to $w_s = 3$ ($w_s = 4$ is illustrated in Table 5.2) to illustrate the trend. Basically, the trend can also be found in other datasets while only the results on Cora are reported since it is the most representative one in this experiment. It is obvious that as the window size increases, the classification performance becomes better for Metapath2vec and DeepWalk. As for HIN2vec, increasing window size also leads to the increasing of meta-path types, so the result does not show a monotonous growth. By displaying the impact of window size on the classification performance, we could infer the importance of long-term information in Cora node classification, which further infers the capability of mSHINE in capturing long-term information.

In summary, the property of capturing long-term information in DeepWalk, HIN2vec and Metapath2vec is controlled by window size (w_s) and the ability of meta-path (aspect) selection is controlled by meta-path length (HIN2vec), incompatibility threshold (AspEm) or even no criteria (Metapath2vec). Compared with them, mSHINE does not require these hyper-parameters which makes mSHINE more

TABLE 5.4: Performance of Node Classification with Cora with Various Window Size from $w_s = 1$ to $w_s = 3$

Metric	Training	DeepWalk			Metapath2vec			HIN2vec		
Selected Aspects		PPP			PPP			PPP		
Window Size		$w_s = 1$	$w_s = 2$	$w_s = 3$	$w_s = 1$	$w_s = 2$	$w_s = 3$	$w_s = 1$	$w_s = 2$	$w_s = 3$
f1-macro	20%	0.3946	0.6845	0.7113	0.5136	0.6411	0.6918	0.7088	0.6870	0.7154
	40%	0.4933	0.7306	0.7491	0.6255	0.6928	0.7234	0.7367	0.7281	0.7445
	60%	0.5401	0.7482	0.7679	0.6696	0.7109	0.7387	0.7495	0.7481	0.7540
	80%	0.5717	0.7577	0.7743	0.6874	0.7239	0.7504	0.7567	0.7542	0.7656
f1-micro	20%	0.6017	0.7592	0.7774	0.6255	0.7206	0.7623	0.7728	0.7635	0.7793
	40%	0.6470	0.7900	0.8040	0.7037	0.7555	0.7821	0.7937	0.7877	0.7978
	60%	0.6681	0.8043	0.8171	0.7354	0.7703	0.7955	0.8025	0.8005	0.8050
	80%	0.6830	0.8090	0.8238	0.7555	0.7800	0.8021	0.8092	0.8050	0.8136

feasible for practical usage besides its state-of-the-art performance as shown in Table 5.2.

5.5.5 Link Prediction

In this section, mSHINE is evaluated via another HIN analysis task which is link prediction. In this experiment, the link prediction is modeled as ranking all possible nodes based on a query node and the performance of link prediction is evaluated through the relevance between the highly ranked nodes and the query node. The related experimental setup will be introduced first which is followed by experimental results and the detailed analysis.

5.5.5.1 Experimental setup

Data preparation. In this experiment, we adopt three widely used datasets from different domains as the experimental datasets which are IMDB, Yelp and DBLP. Following [28], to construct link prediction datasets, for each dataset, we randomly remove 20% of a certain type of edges and the edge types which are selected to remove in DBLP, IMDB and Yelp are set as $P-A$, $U-M$ and $B-U$ respectively. Thus, the experimental datasets are constructed.

Evaluation metrics. After removing edges, network representation learning algorithms are applied to learn HIN node representations based on the remained HIN. The purpose of link prediction is to reconstruct those removed edges. In mSHINE, the prediction objective function given in Equation (5.16), can be used to rank all

possible nodes based on the prediction value. Here, all possible nodes means the nodes with the same type as the target nodes. Since mSHINE is capable of simultaneously capturing information learned from different meta-paths, we will calculate the average relevance based on multiple meta-paths and use this average value to rank nodes. For example, in DBLP, to predict the related authors(A) of a given paper(P), we will calculate the prediction value based on different meta-paths and different three-element node sequence including $V-P \rightarrow A$, $T-P \rightarrow A$ and $A-P \rightarrow A$, then the average value from these meta-paths is used to rank all authors in the HIN. In this case, the well trained node representation learning module which is introduced in Section 5.4.1 can be easily applied in this link prediction task. Top-k recall, top-k precision, Mean Average Precision(MAP) as well as Mean reciprocal rank (MRR) are chosen as metrics for link prediction evaluation [99] here. and the four metrics are given below.

a) *top-k precision*:

$$Pre@k = \frac{\text{num. of hits @}k}{k}$$

b) *top-k recall*:

$$Rec@k = \frac{\text{num. of hits @}k}{\text{num. of relevant nodes}}$$

c) *MAP*:

$$AP@k(V_i) = \frac{1}{k} \sum_{j=1}^k Pre@j \cdot relevant@j$$

where $relevant@j = 1$ if the j th recommended node is relevant otherwise $relevant@j = 0$ and:

$$MAP = \frac{1}{N} \sum_{i=1}^N AP@k(V_i)$$

where N represents the number of all query nodes.

d) *MRR*:

$$MRR = \frac{1}{N} \sum_{i=1}^N \frac{1}{rank_{V_i}}$$

where $rank_{V_i}$ refers to the rank position of the first relevant node in recommendation list when the query node is V_i . MAP gives a general measure of the link prediction quality while MRR only cares about the highest-ranked relevant item.

TABLE 5.5: Performance of Link Prediction

Dataset	Metric		DeepWalk	LINE	Metapath2vec	AspEm	HIN2vec	mSHINE
Selected Aspect					PAP	CONCAT	PA	
DBLP	Pre@k	k=1	0.0690	0.0039	0.0648	0.1694*	0.0392	0.1867
		k=3	0.0438	0.0032	0.0610	0.1055*	0.0326	0.1100
		k=10	0.0221	0.0024	0.0429	0.0496*	0.0200	0.0501
	Rec@k	k=1	0.0587	0.0033	0.0553	0.1483*	0.0342	0.1620
		k=3	0.1095	0.0080	0.1573	0.2702*	0.0824	0.2791
		k=10	0.1831	0.0194	0.3701	0.4179*	0.1669	0.4179
	MAP		0.1045	0.0112	0.1470	0.2415*	0.0803	0.2550
	MRR		0.1149	0.0130	0.1152	0.2577*	0.0870	0.2763
Selected Aspect					UMAMU	CONCAT	UM	
IMDB	Pre@k	k=1	0.0021	0.0043	0.0224	0.2145	0.3202*	0.3180
		k=3	0.0025	0.0039	0.0164	0.2166	0.2832*	0.2942
		k=10	0.0038	0.0041	0.0161	0.1892	0.2344*	0.2334
	Rec@k	k=1	0.0001	0.0002	0.0025	0.0222	0.0335*	0.0370
		k=3	0.0003	0.0006	0.0093	0.0644	0.0867*	0.0917
		k=10	0.0031	0.0027	0.0288	0.1729	0.2257*	0.2186
	MAP		0.0229	0.0126	0.0249	0.1720	0.2195*	0.2212
	MRR		0.0253	0.0226	0.0405	0.3984	0.4981*	0.4981
Selected Aspect					UBCiBU	CONCAT	BU	
YELP	Pre@k	k=1	0.0033	0.0072	0.0145*	0.0134	0.0048	0.0180
		k=3	0.0028	0.0066	0.0116*	0.0111	0.0039	0.0158
		k=10	0.0023	0.0041	0.0079	0.0087*	0.0031	0.0131
	Rec@k	k=1	0.0008	0.0013	0.0005	0.0028*	0.0018	0.0048
		k=3	0.0017	0.0040	0.0012	0.0066*	0.0040	0.0121
		k=10	0.0040	0.0081	0.0027	0.0155*	0.0097	0.0297
	MAP		0.0030	0.0045	0.0022	0.0096*	0.0077	0.0169
	MRR		0.0083	0.0129	0.0219	0.0248*	0.0131	0.0354

As for other baseline algorithms, we also make use of the objective function proposed in respective paper to measure the relevance between nodes. It is worth noting that the objective function value of HIN2vec is calculated based on specific meta-paths and this requires us specifying the meta-path type used for conducting link prediction. So, we choose to use the same meta-path as the removed edge type for each dataset. For example, the removed edge type in DBLP is $P-A$, so the meta-path type we choose to measure the relevance between nodes in DBLP for HIN2vec is $P-A$.

5.5.5.2 Experimental result and discussion

The performance of each algorithm in link prediction task is shown in Table 5.5. The same as Section 5.5.4.2, the best performance of the existing algorithms are marked with * and we highlight the best result for each dataset in bold.

mSHINE outperforms all the state-of-the-art algorithms except HIN2vec when the test data is IMDB. Even comparing with HIN2vec with IMDB, mSHINE achieves top 2 and has comparable performance with HIN2vec. Besides, the effectiveness of the proposed method is quite obvious especially when compared with those algorithms which are designed for the homogeneous information networks and this also indicates the advantages of separately embedding different semantic or structural information in HINs. This is because that the purpose of link prediction is to predict a specific type of edges while algorithms such as DeepWalk and LINE fail to distinguish different edge types and embedding all types of edge information into one embedding might bring in too much noise which is harmful to link prediction performance. However, even different semantic information is separately embedded, the way to select and merge different semantic information learned from different meta-paths(aspects) also influence link prediction performance. We take DBLP and IMDB as the examples to show the influence of meta-paths(aspect) selection in link prediction and the experimental results are shown in Table 5.6. We highlight the best meta-path or meta-path combination for each algorithms in bold. As for Metapath2vec, it usually gets better performance when combining more meta-paths information while the training cost is huge since it requires re-training for each meta-path separately. Regarding HIN2vec, it is capable of simultaneously learning multiple meta-path embeddings which reduces the cost of training. However, combining representations learned from different meta-paths does not ensure that the performance of HIN2vec in link prediction is improved. To obtain the optimal performance of HIN2vec, more meta-path combinations need to be tested while this will increase the cost of testing. On the contrary, we can find from Table 5.5 that mSHINE is able to get satisfied performance by directly combining node representations learned from all different meta-paths so that no more meta-path combination testing is required. Again, compared with other baseline algorithms, the proposed method requires a smaller number of hyper-parameters while achieves a better performance on almost all datasets which shows the effectiveness and feasibility of mSHINE in practical applications.

TABLE 5.6: Performance of Link Prediction Based on Different Meta-path (Aspects)

Dataset	Metric		Metapath2vec			HIN2vec	
Selected Aspect			APTPA	APVPA	APA +APTPA+APVP	PA	PA+PTPA +PAPA+PVPA
DBLP	Pre@k	k=1	0.0690	0.0120	0.1781	0.0392	0.1167
		k=3	0.0408	0.0083	0.1173	0.0326	0.0671
		k=10	0.0229	0.0050	0.0553	0.0200	0.0320
	Rec@k	k=1	0.0564	0.0103	0.1563	0.0342	0.0982
		k=3	0.1059	0.0218	0.3045	0.0824	0.1672
		k=10	0.1967	0.0447	0.4749	0.1669	0.2634
	MAP		0.1033	0.0229	0.2600	0.0803	0.1581
	MRR		0.0751	0.0178	0.1793	0.0870	0.1775
Selected Aspect			MUM	UMDMU	MUM+UMDMU +UMGMU+UMAMU	UM	UM +UMUM+UMAM +UMGM+UMDM
IMDB	Pre@k	k=1	0.0149	0.0043	0.0202	0.3202	0.2006
		k=3	0.0139	0.0068	0.0171	0.2832	0.1572
		k=10	0.0127	0.0086	0.0186	0.2344	0.1249
	Rec@k	k=1	0.0007	0.0022	0.0032	0.0335	0.0218
		k=3	0.0032	0.0052	0.0071	0.0867	0.0446
		k=10	0.0181	0.0144	0.0318	0.2257	0.1093
	MAP		0.0215	0.0223	0.0298	0.2195	0.1152
	MRR		0.0359	0.0280	0.0426	0.4981	0.3425

5.6 Conclusion

In this chapter, we presented a novel HIN representation learning framework which consists of a set of criteria for initial meta-path selection and an HIN representation learning module. The proposed criteria for initial meta-path selection is helpful to reduce the cost of meta-path exploration during training. The representation learning module inspired by basic RNN structure is designed to be capable of simultaneously learning multiple node representations for different meta-paths which improves the learning efficiency especially when multiple meta-paths(aspects) need to be trained. Besides, the learned node representation learning module can be easily applied in downstream link prediction tasks. Through our comprehensive experiments on 5 real-world datasets, we demonstrates the capability of mSHINE in both capturing long-term information as well as efficiently selecting suitable meta-paths for downstream applications without any prior knowledge from experts. In these experiments, the proposed method outperforms the stat-of-the-art methods on most of the datasets in both node classification and link prediction tasks. Besides, by storing state node representations during training, compared with other random-walk-based algorithms, the proposed method is able to capture long-term node information without preparing a huge number of long node sequences for training

which makes mSHINE more efficient. All the training code including the code to find all initial meta-paths is available at <https://github.com/XinyiZ001/mSHINE>.

Chapter 6

eHIN: An Efficient Meta-path Sharing Heterogeneous Information Network Representation Learning Method

6.1 Introduction

When processing HIN, in addition to the information incompatibility which is introduced in detail in Chapter 5, learning efficiency is another challenge for HIN representation learning which is the main focus of this chapter. The learning efficiency can be considered from three perspectives:

1. **Spatial Information.** Known as the topological information, the spatial information of each node is critical to describe the corresponding node. However, most graph embedding methods proposed to process this spatial information by converting them into sequential information which: 1) requires a large number of node sequences to be generated around each node [22, 28]; or 2) recurrently apply the sequential learning unit (i.e., RNN) to each node (e.g., mSHINE) and these methods are inefficient in the perspective of capturing spatial information. On the contrary, most GNN-based methods handle

this well. So, making use of the convolution techniques from GNN-based methods is a feasible direction to address this problem.

2. **Sequential Information.** Meta-path is a sequential combination of node/edge types, so, applying meta-path to handle the heterogeneity requires the methods have the ability to model the sequential information introduced by meta-paths. However, when modeling a long meta-path with GNNs, the memory consumption of each round of parameter update may be large. Even when mini-batch update [25] is applied, the neighborhood size still grows exponentially with the number of GNN layers and this is inefficient in terms of training space. So, in practical use, the number of GNN layers is constricted which leads to a limitation on the length of meta-paths that can be processed.
3. **Shared Information.** The semantic information provided by different meta-paths (aspects) is not completely independent of each other. However, some methods [22, 49] suggest completely separately embedding nodes within individual aspects and then testing each aspect or aspect combination using downstream tasks. This is not efficient for training when a large number of cases need to be examined. Besides, the information shared among aspects is completely ignored, this is harmful to the effectiveness of the result node embeddings. However, the shared node representations technique proposed in mSHINE can well address this problem to some extent.

To address the problems mentioned above, we propose eHIN by fusing the techniques from GNN-based methods (Chapter 4) and graph embedding methods (Chapter 5). Specifically, given a HIN as well as a set of meta-paths, eHIN makes use of the convolution-based unit cooperating with the attention mechanism to simultaneously capture the topological information around nodes as well as the sequential information introduced by meta-paths. Based on such design, we can generate the temporal node representations, which are then considered as the ground truth for training the corresponding static meta-path-based node representations. Multiple carefully designed objective functions are proposed and applied to train node representations by preserving the connectivity proximity of the given HIN and making use of node label information. After training, each static meta-path-based node representations can be easily generated by combining the well-trained shared node representation and the corresponding view generation function, which

simplifies its use in the downstream applications. The main contributions of this work are summarized as follows:

- Proposed eHIN, an efficient novel HIN representation learning method with carefully designed optimization functions making good use of node labels and meta-path, whose effectiveness is verified via comprehensive experiments study (i.e., node classification and link prediction) on 5 real-world datasets.
- Proposed a new mechanism to simultaneously capture both spatial and meta-path-based sequential information when generating the temporal node representations, which are then used to iteratively update the static node representations, so that the captured spatial and meta-path-based sequential information can be integrated into the static node representations. This will simplify the use of node embeddings in downstream applications.
- Proposed shared node representations to facilitate the compatible information across different meta-paths to be shared, which improves the learning efficiency and representation quality, and this is verified through experiments.
- Applied convolution-based unit and meta-path-based non-sampling optimization methods to capture the spatial information which further improved the learning efficiency.

The rest of this chapter is organized as follows. Section 6.2 gives notations used in this paper and some necessary concepts/methods which are used in HIN-related works or our method. The details of the proposed method are described in Section 6.3. Experiments are conducted and detailed analyses are illustrated in Section 6.4. Finally, the conclusion is given in Section 6.5.

6.2 Preliminaries

In this section, we introduce related concepts and notations. As some of the necessary HIN related concepts are already introduced in Section 3.1 and Section 5.2.1, we only introduce the critical concepts which is used in this chapter. Here, we also rephrase the meta-path selection criteria proposed in mSHINE for easier reading of this chapter.

FIGURE 6.1: An example of HIN and network schema. a) An example of CORA information network and a_i, p_i, t_i denote Authors, Papers and terms respectively. b) The network schema of CORA. Three types of nodes and edges are displayed where A, P, T denote the node type of Author, Paper and Term respectively.

TABLE 6.1: An Example of the Meta-path Selection Result

Network Schema	$Rel^{\mathcal{G}}$	$\mathbb{M}$
	$\{Rel_1 : A \rightarrow P \rightarrow P, Rel_2 : A \rightarrow P \rightarrow T,$ $Rel_3 : P \rightarrow P \rightarrow P, Rel_4 : P \rightarrow P \rightarrow A,$ $Rel_5 : P \rightarrow P \rightarrow T, Rel_6 : P \rightarrow A \rightarrow P,$ $Rel_7 : P \rightarrow T \rightarrow P, Rel_8 : T \rightarrow P \rightarrow A,$ $Rel_9 : T \rightarrow P \rightarrow P, Rel_{10} : A \rightarrow P \rightarrow A,$ $Rel_{11} : T \rightarrow P \rightarrow T \}$	$Rel^{M_1} : \{Rel_2, Rel_7, Rel_8, Rel_6\}(APTPA)$ $Rel^{M_2} : \{Rel_9, Rel_5, Rel_7\}(TPPT)$ $Rel^{M_3} : \{Rel_{10}, Rel_6\}(APA)$ $Rel^{M_4} : \{Rel_{11}, Rel_7\}(TPT)$ $Rel^{M_5} : \{Rel_1, Rel_4, Rel_6\}(APPA)$ $Rel^{M_6} : \{Rel_3\}(PPP)$

1. **Meta-path Selection From mSHINE.** According to the definitions of meta-path and meta-path-based random walk (Section 5.2.1), the number of meta-paths in $\mathcal{G}$ is unlimited. It is unrealistic to directly use all meta-paths in practical applications. So, meta-path selection becomes necessary. We choose to directly use the selection criteria proposed by mSHINE(Chapter 5) for meta-path selection, which has been proofed to be effective. Specifically, based on different symmetric meta-paths (only symmetric meta-paths are considered in mSHINE), a large number of node sequences of any length can be generated. However, when we decompose these node sequences into triple sequence type sets, many of the resulted sets are equal. By regarding the meta-paths that result in the same triple sequence type sets as the same, the number of possible meta-paths in $\mathcal{G}$ becomes limited, and all of them are selected as valid meta-paths for training which is denoted as $\mathbb{M} = \{M_1, M_2, \dots, M_c\}$. Besides, each meta-path M_j can be represented with its decomposed triple sequence type set $Rel^{M_j} = \{T_1^v \rightarrow T_2^v \rightarrow T_3^v, T_2^v \rightarrow T_3^v \rightarrow T_4^v, \dots, T_{l-1}^v \rightarrow T_l^v \rightarrow T_{l+1}^v\}$ ¹ and each triple sequence type is denoted as $Rel_r :$

¹In most datasets, the edge type T_i^e can be determined by the node types T_i^v and T_{i+1}^v at its ends, so we eliminate the edge type here for simplification

$T_p^v \rightarrow T_c^v \rightarrow T_t^v$. Given a HIN $\mathcal{G}$, all the possible triple sequence types can be found over the network schema $\mathcal{T}_{\mathcal{G}}$ and form a set that is associated with the whole HIN and it is denoted as $Rel^{\mathcal{G}}$. Table 6.1 shows an example that lists the details of $Rel^{\mathcal{G}}$ and $\mathbb{M}$ which are formed from the given network schema where each meta-path M_j is represented with its decomposed triple sequence type set Rel^{M_j} .

2. **Meta-path-based Neighbors.** Given a triple sequence type $Rel_r : T_p^v \rightarrow T_c^v \rightarrow T_t^v \in Rel^{M_j}$ where M_j is the current meta-path and a center node n_c where $\phi(n_c) = T_c^v$, the meta-path-based neighbors of n_c can be defined as $\mathcal{N}^{T_p^v}(n_c)$ and $\mathcal{N}^{T_t^v}(n_c)$. Here, $\mathcal{N}^{T_p^v}(n_c)$ and $\mathcal{N}^{T_t^v}(n_c)$ indicates n_c 's neighbor nodes with the node type of T_p^v and T_t^v respectively. Considering the directionality integrated in Rel_r , $\mathcal{N}^{T_p^v}(n_c)/\mathcal{N}^{T_t^v}(n_c)$ are also referred to as the *head neighbors/tail neighbors* of n_c respectively in the rest of the paper.

6.3 Methodology

Given a HIN $\mathcal{G} = (\mathbb{V}, \mathbb{E})$ and a set of meta-paths $\mathbb{M} = \{M_1, M_2, \dots, M_c\}$ which are selected based on the meta-path selection criteria from mSHINE, eHIN intends to learn a set of meta-path-based node representation matrices $\{\mathbf{H}^{(i)} \in R^{|\mathbb{V}| \times d}\}_{i=1}^c$, where the important information of $\mathcal{G}$ extracted from each meta-path (M_j) should be well preserved in the corresponding meta-path-based node representation matrix ($\mathbf{H}^{(j)}$) and it can be easily applied to the downstream graph-related applications. The details about how to train $\{\mathbf{H}^{(i)}\}_{i=1}^c$ and generate it for downstream application using are introduced below.

We first present the training process of this framework which consists of three phases: 1) **Training Data Preparation** 2) **Node Representation Generation** and 3) **Objective Optimization**; subsequently, we discuss the details of each phase in the following subsections. At last, we introduce how to generate $\{\mathbf{H}^{(i)}\}_{i=1}^c$ for downstream tasks.

FIGURE 6.2: Overview of eHIN where different node types are denoted with different colors. a) At each iteration, based on the meta-path information (i.e., the current meta-path (M_1), the corresponding triple sequence type ($Rel_2: A \rightarrow P \rightarrow T$) and the negative meta-path (M_3)), the training samples can be prepared accordingly: the center node (p_1), the corresponding head neighbors ($\mathcal{N}^A(p_1) = \{a_1, a_2\}$) and tail neighbors ($\mathcal{N}^T(p_1) = \{t_1, t_2\}$). b) Before generating the temporal node representation of p_1 , the attribute node representation $\vec{h}_{p_1}^s$ of p_1 and the static node representations $\vec{h}_{a_1}^s, \{\vec{h}_{a_2}^s\}$ of $p_1, \mathcal{N}^A(p_1) = \{a_1, a_2\}$ should be prepared through transforming as well as looking up the static node representation matrix (H^s) respectively. In this figure, we use two red vectors to represent the generated $\{\vec{h}_{a_1}^s, \vec{h}_{a_2}^s\}$. Then the temporal node representation $\vec{h}_{p_1, A \rightarrow P}^{(1)}$ is generated based on $\vec{h}_{p_1}^s$ together with $\vec{h}_{p_1}^s$ and $\{\vec{h}_{a_1}^s, \vec{h}_{a_2}^s\}$ (Eq.6.3). $\vec{h}_{p_1, A \rightarrow P}^{(1)}$ is then considered as the ground-truth for updating the static node representation of p_1 ($\mathcal{L}_1^{M_1}$; Eq.6.12); c) In the stage of optimization, the temporal and static node representations of p_1 are used to predict the tail neighbors ($\mathcal{L}_2^{M_1}$; Eq.6.16) and labels ($\mathcal{L}_4^{M_1}$; Eq.6.23). To achieve this, the static node representations ($\{\vec{h}_{t_1}^s, \vec{h}_{t_2}^s\}$) of $\{t_1, t_2\}$ should be prepared respectively in advance which are noted with two green vectors here in the figure. Concurrently, the prediction value of the connection between p_1 and its tail neighbors ($\{t_1, t_2\}$) which is calculated from the negative meta-path (M_3) should be minimized and the corresponding loss is $\mathcal{L}_3^{M_1}$ (Eq.6.22).

6.3.1 Overall Framework

The overall structure of eHIN is shown in Figure 6.2. Specifically, at each training iteration, given the current meta-path information, training samples can be prepared accordingly, based on which, Node Representation Generation module and Optimization module are sequentially applied to train the whole model. In the end, the important information of the HIN should be well preserved in $\mathbf{H}^s$ and $\{\mathcal{F}_i(\cdot)\}_{i=1}^c$.

Training Data Preparation. Given the current meta-path information (i.e., M_j and $Rel_r: T_p^v \rightarrow T_c^v \rightarrow T_t^v \in Rel^{M_j}$), the training sample we process in this work is denoted as $(\mathcal{N}^{T_p^v}(n_c), n_c, n_t, Rel_r, M_j)$ where n_c is the center node, based on which, we prepare the head neighbors $\mathcal{N}^{T_p^v}(n_c)$ of n_c under the given triple sequence type Rel_r . n_t is a random sampled node and the connectivity between n_c and n_t is the proximity that we design to preserve during training.

Node Representation Generation. In this module, two types of node representations will be generated for each node (n_c): *temporal node representation* and *static node representation*. The generation process of the *temporal node representation* aims to integrate rich while heterogeneous neighbor information, which includes two perspectives: spatial information and sequential information. Spatial information, known as the topological information around each node, has been widely discussed in the conventional GNNs [25, 64], where pooling/aggregation functions are usually applied to capture such information. Sequential information here indicates the node/edge type information introduced by meta-path which is critical for HIN representation learning and we choose to use sequential data processing modules (e.g., RNN, GRU, etc.) to capture such information. It is noted that *temporal node representation* is generated based on the neighbor nodes, however, due to the resource limitation, the number of neighbor nodes considered at each iteration is limited, which might lead to the case where only a part of neighbors are considered when generating the *temporal node representation* and this is sub-optimal for use in the downstream applications. Besides, dynamically generating *temporal node representation* each time for practical use usually is not efficient. So, to address the problems existing in *temporal node representation*, *static node representation* is proposed in this work which serves as the steady-state of the corresponding *temporal node representation*. The generation of *static node representation* is much easier by looking up the shared representation matrix ($\mathbf{H}^s$)

and the corresponding view generation functions ($\{\mathcal{F}_j(\cdot)\}_{j=1}^c$). At last, in some cases where the node attributes are available, we will first generate node attribute representations based on these node attributes and then integrate them into the temporal/static node representations.

Optimization. During optimization, the static node representations will be trained to approximate the newly generated temporal node representation so that the rich information captured by the temporal node representations could be preserved in static node representations. Besides, multiple objective functions are designed to optimize the newly generated temporal node representations so that the connectivity information from different meta-paths could be well modeled. Lastly, we also designed an optimization objective to integrate label information into node representations. Each related part is introduced in detail below.

6.3.2 Node Representation Generation

6.3.2.1 Node Attribute Representation (Optional)

To integrate the rich node attribute information (if available), node attribute representation is firstly generated by transforming different types of node attributes into a shared latent representation space, which is then used to generate the temporal/static node representations. This can be written as:

$$\vec{x}_{n_c} = W_x^{(\phi(n_c))} \vec{x}'_{n_c} \quad (6.1)$$

where $\vec{x}'_{n_c} \in \mathbb{R}^{d \times 1}$ denotes the attributes vector of the center node n_c . $W_x^{(\phi(n_c))} \in \mathbb{R}^{d \times d}$ is a trainable transform matrix which is associated with the node type $\phi(n_c)$.

6.3.2.2 Static Meta-path-based Node Representation

Static meta-path-based node representation $\vec{h}_{n_c}^{(j)}$, known as the static node representation introduced above, can be easily generated by combining $\mathbf{H}^s$ and $\{\mathcal{F}_i(\cdot)\}_{i=1}^c$. This specific process can be represented as:

$$\begin{aligned} \vec{h}_{n_c}^{(j)} &= \mathcal{F}_j(\vec{h}_{n_c}^s) \\ &= f((\vec{h}_{n_c}^s \circ \vec{u}_{h, \phi(n_c)}^{(j)})^T W_{state}) \end{aligned} \quad (6.2)$$

where $\vec{h}_{n_c}^s \in \mathbb{R}^{d \times 1}$ is a row of $\mathbf{H}^s$ which represents the shared node representation of n_c and it is randomly initialized before training. $\mathcal{F}_j(\cdot)$ is a view generation function associated with the meta-path M_j . $\vec{u}_{h,\phi(n_c)}^{(j)} \in \mathbb{R}^{d \times 1}$ denotes a trainable vector integrated in $\mathcal{F}_j(\cdot)$ which serves as a gating to filter meta-path M_j related information from $\vec{h}_{n_c}^s$. The value of $\vec{u}_{h,\phi(n_c)}^{(j)}$ is regularized within -1 and 1 . W_{state} represents a trainable transform matrix and we choose to use $f(\cdot) = \tanh(\cdot)$ here.

6.3.2.3 Temporal Meta-path-based Node Representation

Given the current meta-path M_j and the triple sequence type $Rel_r: T_p^v \rightarrow \phi(n_c) \rightarrow T_t^v$, the generation of the temporal meta-path-based node representation, which is known as the temporal node representation above, can be formally written as:

$$\tilde{h}_{n_c, (T_p^v \rightarrow \phi(n_c))}^{(j)} = f_{pool}(\{f_{trans}(\vec{x}_{n_c}, \vec{h}_{n_k}^{(j)}), \forall n_k \in \mathcal{N}^{T_p^v}(n_c)\}) \quad (6.3)$$

where $f_{trans}(\cdot)$ is a sequential data processing model (we choose to use GRU [97] here) and it aims to model the sequential node type information from n_k to n_c . $f_{trans}(\cdot)$ can be formally written as:

$$\vec{z} = \sigma(W_z \vec{x}_{n_c} + U_z \vec{h}_{n_k}^{(j)} + \vec{b}_z) \quad (6.4)$$

$$\vec{r} = \sigma(W_r \vec{x}_{n_c} + U_r \vec{h}_{n_k}^{(j)} + \vec{b}_r) \quad (6.5)$$

$$\vec{h} = \tanh(W_h \vec{x}_{n_c} + U_h \vec{h}_{n_k}^{(j)} \circ \vec{r}) \quad (6.6)$$

$$f_{trans}(\vec{x}_{n_c}, \vec{h}_{n_k}^{(j)}) = \vec{z} \circ \vec{h}_{n_k}^{(j)} + (1 - \vec{z}) \circ \vec{h} \quad (6.7)$$

where $n_k \in \mathcal{N}^{T_p^v}(n_c)$. This indicates the sequential node type information that is currently modeled is constricted as $T_p^v \rightarrow \phi(n_c)$ by Rel_r .

$f_{pool}(\cdot)$ is used to model the spatial information around n_c by pooling the captured information from its neighbor nodes ($\mathcal{N}^{T_p^v}(n_c)$). As there is no natural ordering in $\mathcal{N}^{T_p^v}(n_c)$, the pooling function should be permutation invariant and we examined two candidate functions here:

1. Mean Pooling:

$$\tilde{h}_{n_c, (T_p^v \rightarrow \phi(n_c))}^{(j)} = \frac{\sum_{n_k \in \mathcal{N}^{T_p^v}(n_c)} f_{trans}(\vec{x}_{n_c}, \vec{h}_{n_k}^{(j)})}{|\mathcal{N}^{T_p^v}(n_c)|} \quad (6.8)$$

2. Attention Pooling:

$$\tilde{h}_{n_c, (T_p^v \rightarrow \phi(n_c))}^{(j)} = \sum_{n_k \in \mathcal{N}^{T_p^v}(n_c)} a_{n_k} \cdot f_{trans}(\vec{x}_{n_c}, \vec{h}_{n_k}^{(j)}) \quad (6.9)$$

where

$$a_{n_k} = \frac{\exp\left(f_s\left(\vec{h}_{n_c}^{(j)}, f_{trans}(\vec{x}_{n_c}, \vec{h}_{n_k}^{(j)})\right)\right)}{\sum_{n_t \in \mathcal{N}^{T_p^v}(n_c)} \exp\left(f_s\left(\vec{h}_{n_c}^{(j)}, f_{trans}(\vec{x}_{n_c}, \vec{h}_{n_t}^{(j)})\right)\right)} \quad (6.10)$$

and

$$f_s\left(\vec{h}_{n_c}^{(j)}, f_{trans}(\vec{x}_{n_c}, \vec{h}_{n_k}^{(j)})\right) = f_{trans}(\vec{x}_{n_c}, \vec{h}_{n_k}^{(j)})^T (\vec{h}_{n_c}^{(j)} + \vec{e}_{(T_p^v \rightarrow T_c^v)}) \quad (6.11)$$

where $\vec{e}_{(T_p^v \rightarrow T_c^v)}$ denotes the edge type embedding which will be introduced in detail in the Section 6.3.3.1.

Note that we use tilde($\sim$) in Equation (6.3) to differentiate two different types of meta-path-based node representations. $\tilde{h}_{n_c}^{(j)}$ with a tilde denotes the temporal meta-path-based node representation which is calculated through $f_{pool}(\cdot)$ and $f_{trans}(\cdot)$ on the fly. This procedure can enrich the information of $\tilde{h}_{n_c}^{(j)}$ through integrating neighbor node information. On the contrary, $\vec{h}_{n_c}^{(j)}$ without the tilde denotes the static meta-path-based node representation which is generated through $\mathcal{F}_j(\cdot)$ and $\vec{h}_{n_c}^s$. This procedure is more like retrieval since there is no information passing from other nodes and we only focus on the center node n_c itself. The general procedure that we generate $\tilde{h}_{n_c}^{(j)}$ and $\vec{h}_{n_c}^{(j)}$ and their relationship is shown in Figure 6.3. In the downstream applications, $\vec{h}_{n_c}^{(j)}$ is used as the final node representation of n_c associated with the meta-path M_j .

FIGURE 6.3: The generation of $\tilde{h}_{n_c}^{(j)}$ and $\vec{h}_{n_c}^{(j)}$ and their relationship

6.3.3 Objective Optimization

The optimization objective we designed in this work includes the objective for: 1) updating the static node representations; 2) integrating the heterogeneous connectivity information; 3) preserving label information.

6.3.3.1 Static Node Representation Updating

As introduced in Section 6.3.2.3, $\vec{h}_{n_c}^{(j)}$ will be used as the final steady node representation of n_c associated with M_j . To integrate richer neighbor spatial and sequential information into $\vec{h}_{n_c}^{(j)}$, we use the dynamically generated $\tilde{\vec{h}}_{n_c}^{(j)}$ to update $\vec{h}_{n_c}^{(j)}$ during optimization and the loss is:

$$\mathcal{L}_1^{M_j}(n_c, Rel_r) = \left(\vec{h}_{n_c}^{(j)} + \vec{e}_{(T_p^v \rightarrow T_c^v)} - \tilde{\vec{h}}_{n_c, (T_p^v \rightarrow T_c^v)}^{(j)} \right)^2 \quad (6.12)$$

It is noted that $\tilde{\vec{h}}_{n_c, (T_p^v \rightarrow T_c^v)}^{(j)}$ is used as the ground truth in Equation (6.12) and it will not be affected during the minimization of $\mathcal{L}_1^{M_j}(n_c, Rel_r)$. $\tilde{\vec{h}}_{n_c, (T_p^v \rightarrow T_c^v)}^{(j)}$ is generated based on the neighbor nodes with the specific node type T_p^v , therefore, the edge type information is naturally integrated. However, when generating $\vec{h}_{n_c}^{(j)}$, the edge type information is not considered. Therefore, an edge type embedding $\vec{e}_{(T_p^v \rightarrow T_c^v)}$ is necessarily introduced here to bridge $\tilde{\vec{h}}_{n_c, (T_p^v \rightarrow T_c^v)}^{(j)}$ and $\vec{h}_{n_c}^{(j)}$. In this work, the edge type embeddings are represented with trainable parameters and are shared by all meta-paths.

6.3.3.2 Heterogeneous Connection Information Preservation

In the field of graph embedding, a common proximity that most algorithms preserve for training is the connectivity. Specifically, a prediction module/function $P(\cdot)$ is firstly defined to evaluate the possibility that a node pair (n_c, n_t) is connected. Then, $P(n_c, n_t)$ will be trained to generate a relatively high prediction value when (n_c, n_t) is connected and relatively low value if (n_c, n_k) is not connected. When the connectivity information of a graph can be well modeled through $P(\cdot)$, it is considered that the important information of this graph is well preserved in the trainable parameters in $P(\cdot)$. In HIN representation learning, not only connectivity information, but the connection type information should also be considered,

and this is the reason why the training sample we process in this work is denoted as $(\mathcal{N}^{T_p^v}(n_c), n_c, n_t, Rel_r, M_j)$, which not only provide node pair information (n_c, n_t) but also the meta-path information (i.e., M_j and $Rel_r: T_p^v \rightarrow \phi(n_c) \rightarrow T_t^v \in Rel^{M_j}$). The polarity of this sample belongs to one of the following cases²:

1. **Connected node pairs under given meta-path.** (n_c, n_t) is connected and $\phi(n_t) = T_t^v$ so that $Rel'_r: T_p^v \rightarrow \phi(n_c) \rightarrow \phi(n_t)$ which is formed by node n_c and n_t equals to the provided Rel_r . This case is considered as a positive sample.
2. **Unconnected node pairs under given meta-path.** $Rel'_r: T_p^v \rightarrow \phi(n_c) \rightarrow \phi(n_t)$ still equals to Rel_r . However, (n_c, n_t) is not connected. This case is considered as a negative sample.
3. **Connected node pairs under wrong meta-path.** (n_c, n_t) is connected while the formed $Rel'_r: T_p^v \rightarrow \phi(n_c) \rightarrow \phi(n_t)$ doesn't equal to Rel_r , which indicates M_j is a wrong meta-path and this case is considered as a negative sample in this case.

The loss for modelling Case 1 can be written as:

$$\mathcal{L}_{pos} = w_{c,t}^{M_j^+} (P_j(n_c, n_t, Rel_r) - R_{c,t}^{M_j})^2 \quad (6.13)$$

Case 2 and 3 can be formally written as:

$$\mathcal{L}_{neg} = w_{c,t}^{M_j^-} (P_j(n_c, n_t, Rel_r) - R_{c,t}^{M_j})^2 \quad (6.14)$$

where the prediction function $P_j(n_c, n_t, Rel_r)$ is defined as:

$$P_j(n_c, n_t, Rel_r) = \frac{1}{2} (\vec{h}_{n_c}^{(j)} + \vec{e}_{(T_p^v \rightarrow T_c^v)} + \tilde{\vec{h}}_{n_c, (T_p^v \rightarrow T_c^v)}^{(j)})^T \vec{h}_{n_t}^{(j)} \quad (6.15)$$

$R_{c,t}^{M_j}$ is the target value of this prediction. It is usually set as a higher value (e.g., 1) for connected node pairs (Case 1) and a lower value (e.g., 0) for unconnected node pairs (Case 2, 3). $w_{c,t}^{M_j}$ denotes the weight of each training sample and the detailed way to set this value can be found in [100].

Normally, under the given meta-path M_j and a triple sequence type $Rel_r: T_p^v \rightarrow \phi(v_c) \rightarrow T_t^v$ where $Rel_r \in Rel^{M_j}$, the possible training samples from case 2 are of

²There is still a fourth case where the node pair is unconnected under wrong meta-path. This case is ignored in this work since it is much easy to be modeled compared with other cases

much larger scale than the samples from case 1, so we choose to use meta-path-based non-sampling method which is inspired from [100] to optimize the loss from case 1 and case 2 simultaneously. The loss can be written as:

$$\mathcal{L}_2^{M_j}(n_c, Rel_r) = \sum_{\substack{n_k \in \mathcal{V}, \\ \phi(n_k)=T_t^v}} w_{c,k}^{M_j} (P_j(n_c, n_k, Rel_r) - R_{c,k}^{M_j})^2 \quad (6.16)$$

Usually the target value $R_{c,t}^{M_j}$ of negative samples are set as 0 and Equation (6.16) can be written as:

$$\begin{aligned} \mathcal{L}_2^{M_j}(n_c, Rel_r) = & \tilde{\mathcal{L}}_{2-pos}^{M_j}(n_c, Rel_r) + \tilde{\mathcal{L}}_{2-all}^{M_j}(n_c, Rel_r) \\ & + \sum_{n_k \in \mathcal{N}^{T_t^v}(n_c)} w_{c,k}^{M_j} \left(R_{c,k}^{M_j}\right)^2 \end{aligned} \quad (6.17)$$

where

$$\begin{aligned} \tilde{\mathcal{L}}_{2-pos}^{M_j}(n_c, Rel_r) = & \sum_{n_k \in \mathcal{N}^{T_t^v}(n_c)} -2w_{c,k}^{M_j^+} P_j(n_c, n_k, Rel_r) R_{c,k}^{M_j} \\ & + \sum_{n_k \in \mathcal{N}^{T_t^v}(n_c)} (w_{c,k}^{M_j^+} - w_{c,k}^{M_j^-}) P_j^2(n_c, n_k, Rel_r) \end{aligned} \quad (6.18)$$

and

$$\tilde{\mathcal{L}}_{2-all}^{M_j}(n_c, Rel_r) = \sum_{\substack{n_k \in \mathcal{V} \\ \phi(n_k)=T_t^v}} w_{c,k}^{M_j^-} P_j^2(n_c, n_k, Rel_r) \quad (6.19)$$

here, Equation (6.19) can be efficiently calculated through:

$$\begin{aligned} \tilde{\mathcal{L}}_{2-all}^{M_j}(n_c, Rel_r) = & \frac{1}{4} \sum_a \sum_b \left(\left(\vec{h}_{n_c,a}^{(j)} + \vec{e}_{(T_p^v \rightarrow T_c^v),a} + \tilde{h}_{n_c,a}^{(j)} \right) \right. \\ & \times \left(\vec{h}_{n_c,b}^{(j)} + \vec{e}_{(T_p^v \rightarrow T_c^v),b} + \tilde{h}_{n_c,b}^{(j)} \right) \\ & \times \left. \left(\sum_{n_k \in \mathcal{V}, \phi(n_k)=T_t^v} w_{c,k}^{M_j^-} \vec{h}_{n_k,a}^{(j)} \vec{h}_{n_k,b}^{(j)} \right) \right) \end{aligned} \quad (6.20)$$

so $\mathcal{L}_2^{M_j}$ could be efficiently calculated by applying the meta-path-based non-sampling optimization method.

As for case 3, we prepare training samples based on the positive samples from case 1. Specifically, given a positive sample $(\mathcal{N}^{T_p^v}(n_c), n_c, n_t, Rel_r, M_j)$ where $Rel_r: T_p^v$

$\rightarrow \phi(n_c) \rightarrow T_t^v \in Rel^{M_j}$, the negative meta-path set of M_j associated with Rel_r is firstly defined as:

$$Neg(M_j, Rel_r) = \{M_k, \forall Rel_i : T_p^v \rightarrow \phi(n_c) \rightarrow T_t' \in Rel^{M_k} \wedge T_i' \neq T_t^v \wedge Rel_i \in Rel^G\} \quad (6.21)$$

Then, we simultaneously model the samples from case 1 and case 3 which is formally written as:

$$\begin{aligned} \mathcal{L}_3^{M_j}(n_c, Rel_r) = & \sum_{n_k \in \mathcal{N}^{T_t^v}(n_c)} \left(P_j(n_c, n_k, Rel_r) - R_{c,k}^{M_j} \right)^2 \\ & + \frac{1}{L} \sum_{M_l \in Neg(M_j, Rel_r)}^{l=1} \left(P_l(n_c, n_k, Rel_r) - R_{c,k}^{M_l} \right)^2 \end{aligned} \quad (6.22)$$

where the value L denotes the number of negative meta-paths randomly selected for each positive sample. It is noted that the negative meta-path set of some meta-paths is empty and we just ignore the corresponding loss in this case.

6.3.3.3 Label Information Preservation

To integrate label information into node representation, we abstract each label as a node of the HIN and associate each label with a randomly initialized representation. Given a node n_c and the label set $\mathcal{Z} = \{z_1, z_2, \dots, z_n\}$, the label related loss is modeled as:

$$\mathcal{L}_4^{M_j}(n_c, Rel_r) = \sum_{z_i \in \mathcal{Z}} (P_j(n_c, z_i, Rel_r) - R_{c,z_i})^2 \quad (6.23)$$

where R_{c,z_i} is used to discriminate if the node n_c is associated with the label z_i .

6.3.3.4 Overall Objective Function

Overall, the optimization objective of the proposed method can be summarized and formally written as:

$$\begin{aligned} \mathcal{L}_{all} = & \sum_{n_c \in \mathcal{V}} \sum_{M_j \in \mathcal{M}} \sum_{Rel_r \in Rel^{M_j}} \left(\mathcal{L}_1^{M_j}(n_c, Rel_r) \right. \\ & \left. + \mathcal{L}_2^{M_j}(n_c, Rel_r) + \mathcal{L}_3^{M_j}(n_c, Rel_r) + \mathcal{L}_4^{M_j}(n_c, Rel_r) \right) \end{aligned} \quad (6.24)$$

then all the trainable parameters in the proposed structure are adjusted to minimize the overall optimization objectives $\mathcal{L}_{all}$ so that the important information of $\mathcal{G}$ could be well preserved in node representations.

The pseudocodes for overall eHIN, including meta-path information preparation, training sample preparation, and training procedure, are presented in Algorithm 3. The subscript attached with each function in Algorithm 3 denotes the corresponding equation number as used above.

6.3.4 Representation Generation for Downstream Tasks

When the training is finished, the static meta-path-based node representations $\mathbf{H}^{(j)}$, which serves as the steady-state of the corresponding temporal node representations, are selected to be used in the downstream tasks. As introduced above, $\mathbf{H}^{(j)}$ can be much easily generated by combining $\mathbf{H}^s$ and $\mathcal{F}_j(\cdot)$, which is formally given by Equation (6.2).

6.4 Experiments

6.4.1 Datasets

In this experiments, we use HINs formed from five real-world public datasets: Douban Movie 2, IMDB-S, CORA (with node features), IMDB-L and Last_FM, to evaluate the performance of the proposed method. The description of each dataset can be found in Section 3.2.

Algorithm 3 eHIN

Input: An information network $\mathcal{G}$, a meta-path set $\mathcal{M}$.
Output: The shared representation matrix $\mathbf{H}^s$, the view generation function set $\mathcal{F}$.

- 1: Initialize $\mathbf{H}^s$, $\mathcal{F}$, and other trainable parameters;
- 2: **while** not converged **do**
- 3: **for** $M_j \in \mathcal{M}$ **do**
- 4: sample a triple sequence type $Rel_r: T_p^v \rightarrow T_c^v \rightarrow T_t^v \in Rel^{M_j}$;
- 5: sample the center node n_c where $\phi(n_c) = T_c^v$;
- 6: prepare head/tail neighbors: $\mathcal{N}^{T_p^v}(n_c)$, $\mathcal{N}^{T_t^v}(n_c)$;
- 7: sample a M_j^- from $Neg(M_j, Rel_r)$;
- 8: **Start** $\triangleright$ prepare static node representation
- 9: $\vec{h}_{n_c}^{(j)} \leftarrow f_{6.2}^j(n_c)$;
- 10: $\vec{h}_{n_c}^{(j-)} \leftarrow f_{6.2}^{j-}(n_c)$;
- 11: **End**
- 12: **Start** $\triangleright$ prepare temporal node representation
- 13: $\{\vec{h}_{n_k}^{(j)}\} \leftarrow \{f_{6.2}^j(n_k), \forall n_k \in \mathcal{N}^{T_p^v}(n_c)\}$;
- 14: $\{\vec{h}_{n_k}^{(j-)}\} \leftarrow \{f_{6.2}^{j-}(n_k), \forall n_k \in \mathcal{N}^{T_p^v}(n_c)\}$;
- 15: $\tilde{h}_{n_c, (T_p^v \rightarrow T_c^v)}^{(j)} \leftarrow f_{6.3}^j(n_c, \{\vec{h}_{n_k}^{(j)}\}, Rel_r)$;
- 16: $\tilde{h}_{n_c, (T_p^v \rightarrow T_c^v)}^{(j-)} \leftarrow f_{6.3}^{j-}(n_c, \{\vec{h}_{n_k}^{(j-)}\}, Rel_r)$;
- 17: **End**
- 18: **Start** $\triangleright$ evaluate sample polarity
- 19: $\{P_j(n_c, n_k, Rel_r)\} \leftarrow \{f_{6.15}^j(n_c, n_k, Rel_r), \forall n_k \in \mathcal{N}^{T_t^v}(n_c)\}$;
- 20: $\{P_{j-}(n_c, n_k, Rel_r)\} \leftarrow \{f_{6.15}^{j-}(n_c, n_k, Rel_r), \forall n_k \in \mathcal{N}^{T_t^v}(n_c)\}$;
- 21: **End**
- 22: **Start** $\triangleright$ calculate overall loss
- 23: $\mathcal{L}_1^{M_j}(n_c, Rel_r) \leftarrow f_{6.12}(\tilde{h}_{n_c, (T_p^v \rightarrow T_c^v)}^{(j)}, \vec{h}_{n_c}^{(j)})$;
- 24: $\mathcal{L}_2^{M_j}(n_c, Rel_r) \leftarrow f_{6.16}(\{P_j(n_c, n_k, Rel_r)\})$;
- 25: $\mathcal{L}_3^{M_j}(n_c, Rel_r) \leftarrow f_{6.22}(\{P_{j-}(n_c, n_k, Rel_r)\})$;
- 26: $\mathcal{L}_4^{M_j}(n_c, Rel_r) \leftarrow f_{6.23}(\tilde{h}_{n_c, (T_p^v \rightarrow T_c^v)}^{(j)}, \vec{h}_{n_c}^{(j)})$;
- 27: **End**
- 28: $\mathcal{L}_{all} \leftarrow f_{6.24}(\mathcal{L}_1^{M_j}, \mathcal{L}_2^{M_j}, \mathcal{L}_3^{M_j}, \mathcal{L}_4^{M_j})$;
- 29: minimize $\mathcal{L}_{all}$ by updating trainable parameters.
- 30: **end for**
- 31: **end while**
- 32: **return** $\mathbf{H}^s$, $\mathcal{F}$;

6.4.2 Methods to Compare

To evaluate the effectiveness of the proposed method, we compare it against 9 baseline methods which comprehensively cover the main existing node representation learning methods. Specifically, we selected GNN-based methods that designed for both homogeneous information networks (i.e., GraphSAGE) and HINs (i.e., MAGNN, HGT, ie-HGCN), as well as graph embedding based methods for both homogeneous information networks (i.e., DeepWalk, LINE) and HINs (i.e., AspEm, mSHINE, HIN2vec). More details of each baseline method are shown below:

GraphSAGE [25] is a state-of-the-art GCN-based method for homogeneous network. It proposed to generate node representations through sampling and aggregating node information from a node’s local neighborhoods.

MAGNN [23] is designed based on HAN [73] where node features are generated through sampling and aggregating their meta-path-based neighbors’ information. However, MAGNN claimed that the information of intermediate nodes along the meta-path should also be preserved.

HGT [72] is designed based on GAT [64] where multi-head attention is applied at each layer based on different edge types. By stacking multiple layers, all possible meta-paths whose length equals to the number of layers are considered.

ie-HGCN [20] applied different levels of attention to generate node features from different meta-paths which is more efficient than HGT. Same as MAGNN and HGT, ie-HGCN considered intermediate nodes information along each meta-path while the length of meta-paths is limited.

DeepWalk [13], **LINE** [15], **HIN2vec** [28], **AspEm** [49] are all graph embedding methods and details can be found in Section 5.5.

mSHINE is detailed introduced in Chapter 5.

6.4.3 Implementation Details

The details of implementation of our experiments are summarized below which are applied to all the datasets.

TABLE 6.2: Statistics of Extracted Meta-paths for Each Dataset

Dataset	Douban Movie 2	IMDB-S	CORA	IMDB-L	Last_FM
Num. of Meta-paths	15	3	6	10	6
Num. of Rel_r	21	6	11	20	9

1. Node index will be considered as the input feature of each node if no node attribute is available. Nodes without attributes are denoted as '–' in Table 3.2.
2. The number of extracted meta-paths as well as the related information is shown in Table 6.2.
3. The number of negative meta-paths L sampled for each positive sample is set as 1. Batch size is set as $B = 128$ for each iteration and the number of epochs is set as $E = 500$ which is the same for all datasets. The dimension of node representations(d) is set as 64.
4. Adam [101] with the learning rate of 0.001 is used as the optimizer in this work.
5. The static meta-path-based node representations are used in experimental studies.
6. Experiments are conducted in Nvidia Tesla P100 Cluster.
7. Other hyper-parameters are the same as other algorithms which will be introduced individually in below.

6.4.4 Node Classification

In this section, we evaluate the effectiveness of eHIN through node classification. First, the relevant experimental settings will be introduced which is followed by the experimental results and the corresponding analysis.

6.4.4.1 Experimental setup

Data preparation. Datasets used in node classification tasks should provide node labels. Therefore, CORA, Douban Movie 2 and IMDB-S are chosen in this work

TABLE 6.3: Performance of Node Classification

Dataset	Metric	Ratio	LINE	Deep-Walk	AspEm	HIN2vec	mSHINE	Graph-Sage	MAGNN	ie-HGCN	HGT	eHIN
Selected Aspect					AMD		AMA					DMD
IMDB-S	Macro	20%	48.00	44.72	48.67	49.49	47.08	<u>60.17</u>	59.74	56.41	56.92	60.61
		40%	50.93	46.55	51.65	52.49	49.60	62.23	61.66	58.25	58.61	<u>62.10</u>
		60%	52.20	47.85	52.53	54.15	50.64	62.34	<u>62.36</u>	58.95	59.05	63.04
		80%	53.08	48.06	53.15	55.16	51.54	62.61	<u>63.11</u>	59.74	59.66	63.30
	Micro	20%	49.14	45.85	49.80	50.56	48.35	<u>60.26</u>	59.68	56.66	56.97	60.77
		40%	52.20	47.86	53.00	53.69	51.25	62.28	61.58	58.42	58.52	<u>62.26</u>
		60%	53.69	49.23	54.01	55.49	52.41	<u>62.43</u>	62.29	59.20	58.93	63.21
		80%	54.72	49.60	54.86	56.68	53.47	62.75	<u>63.02</u>	60.13	59.69	63.54
Selected Aspect					AMU		MUUM				DMUMD	
Douban Movie 2	Macro	20%	14.96	14.96	16.64	16.62	<u>17.17</u>	1.67	1.68	8.37	4.59	17.71
		40%	14.10	14.10	16.38	16.75	<u>17.53</u>	1.67	1.82	7.20	3.40	17.78
		60%	13.36	13.36	15.71	16.15	16.91	1.66	1.87	6.60	2.99	<u>16.52</u>
		80%	12.97	12.97	15.31	<u>15.62</u>	15.49	1.66	1.88	6.37	2.86	16.80
	Micro	20%	43.10	43.10	45.68	47.66	45.70	24.08	26.73	33.33	25.36	<u>47.40</u>
		40%	45.82	45.82	49.13	<u>50.23</u>	48.74	23.80	29.36	33.33	25.12	50.51
		60%	46.21	46.21	49.85	50.92	49.58	23.45	30.66	33.36	24.86	<u>50.80</u>
		80%	46.43	46.43	50.08	<u>51.03</u>	49.59	23.50	31.07	33.31	24.93	51.58
Selected Aspect					CONCAT		PPP				PPP	
CORA	Macro	20%	64.10	64.44	63.64	68.34	63.00	55.41	<u>71.82</u>	68.04	52.63	73.04
		40%	67.42	67.50	66.74	71.18	65.16	57.10	<u>71.95</u>	69.29	53.46	74.47
		60%	68.47	68.37	67.55	71.87	65.91	57.27	<u>72.50</u>	69.70	53.52	74.93
		80%	69.08	68.85	67.44	72.51	66.00	57.75	<u>73.46</u>	70.09	53.65	74.94
	Micro	20%	72.38	72.56	72.56	75.48	71.64	66.83	<u>77.43</u>	76.05	64.17	78.86
		40%	74.69	74.84	74.95	77.58	73.23	68.08	<u>77.60</u>	76.83	64.73	79.90
		60%	75.46	75.58	75.65	<u>78.04</u>	73.89	68.25	77.85	76.88	64.86	80.10
		80%	75.85	75.88	75.77	78.54	74.10	68.45	<u>78.67</u>	77.11	65.07	80.57

to evaluate the performance and more details can be found in Section 3.2. It is noted that, different from the datasets used in mSHINE where all node attributes are ignored, the node attributes (if available) should be preserved in this work.

Training. During the training procedure, for each dataset, we randomly split the nodes with labels into training(20%), validation(40%) and testing(40%) sets. For semi-supervised learning models(i.e. GraphSAGE, MAGNN, HGT, ie-HGCN), the training set is used for updating parameters and early stopping is applied based on the validation set. At last, the node embeddings of testing sets generated from each model will be fed to SVM(Linear)³ to evaluate the performance. For unsupervised learning models(i.e. DeepWalk, LINE, HIN2vec, AspEm, mSHINE), all the nodes are used for training while only the node embeddings of the testing set will be fed to SVM. Since eHIN is considered as an unsupervised learning method, so all the nodes will be applied when we train eHIN. However, it is noted that when we train eHIN based on $\mathcal{L}_4^{M_j}$ where the label information is integrated, only the label information of the nodes from the training set will be used.

Evaluation. We use the classification results on linear SVM to evaluate the performance of the learned node representations from each model. Following the

³<https://scikit-learn.org/stable/>

experimental setup of [4, 73], we set four training ratios($\{20\%, 40\%, 60\%, 80\%\}$). At each ratio, we will randomly divide the node representations generated from the testing set mentioned above into new training set and testing set according to the training ratio. The classification results from SVM of the new testing set will be used to evaluate the corresponding model [23]. For a fair comparison, we will repeat this procedure for 10 times and the average results will be reported. As for the result, we use *f1-macro* and *f1-micro* score to evaluate each model [99].

Hyper-parameter setting. For random-walk-based algorithms(i.e. DeepWalk and HIN2vec), the number of node sequences extracted from each node w_n and the node sequence length w_l is set as the same as DeepWalk [13] where $w_n = 80$ and $w_l = 40$. The window size has effect on the number of meta-paths for HIN2vec, so we choose to follow the setting of HIN2vec where $w_s = 4$. Besides, the negative sampling ratio is set as 5. As for GCN-based models such as GraphSAGE, HGT, MAGNN and ie-HGCN, the sampling width is unifiedly set as 64 while other hyper-parameters are the same as reported in the original paper. As for methods which are designed for homogeneous network such as GraphSAGE, DeepWalk and LINE, node/edge types will be ignored in our experiments. Besides, GraphSAGE has mentioned multiple candidate aggregators in the original paper and we examined three of them: mean, meanpooling and maxpooling and the best results will be reported. In AspEm, we firstly select valid aspects by following the default steps, then node representations are learned within each aspects separately. Here, LINE is chosen for AspEm because its performance is better than PTE which is suggested in the original paper. For AspEm, mSHINE and eHIN, the best result among all possible aspects(meta-paths) or their combinations will be reported.

6.4.4.2 Experimental results and discussion

The classification result is shown in Table 6.3 where the best performance of each dataset is highlighted in bold and we also marked the second-best performance with underscore for convenient comparison.

eHIN achieves the state-of-the-art or comparable performance across different training ratio and datasets which indicates the effectiveness of the proposed method in node classification. On IMDB-S, all the GNN-based algorithms achieve better

performance compared with graph embedding methods and this illustrates the importance of the node attributes as well as label information for node classification in IMDB-S. The good performance of eHIN in IMDB-S shows its ability of integrating useful node attributes and label information into node embeddings. On the other hand, when processing Douban Movie where no node attributes are available, all the GNN-based algorithms have much worse performance than graph embedding methods. This indicates that the performance of these GNN-based algorithms highly relies on the quality of the initialization of node features. However, unlike normal GNN-based methods which are simply trained based on label information, eHIN also simultaneously model the heterogeneous connection information which is actually more similar to the graph embedding methods and this allows eHIN handles datasets without node attributes well. As for CORA where both node attributes and connection information are important, eHIN shows its superiority of combining the information extracted from different perspectives and it consistently outperforms other methods. Although there is no evidence that all the HIN-based methods are absolutely superior to methods which are designed for homogeneous information networks, it is still easy to find that the best performance for all datasets is achieved by HIN-based methods except IMDB-S where node attributes dominate the performance.

6.4.5 Link Prediction

In this section, we evaluate the proposed method with link prediction task. The way modelling the link prediction task is the same as Section 5.5.5. More experimental settings and results as well as the corresponding analysis are shown below.

6.4.5.1 Experimental setup

Data preparation. IMDB-L, Last_FM, Douban Movie and CORA are chosen for link prediction experiment. To construct link prediction datasets, 20% of a certain type of edges are removed from each dataset and these removed edges are considered as the ground truth for link prediction⁴, then the node representations are learned based on the remained HIN. For IMDB-L, Last_FM, Douban Movie

⁴When we remove edges, we ensure that no isolated nodes are generated

TABLE 6.4: Performance of Link Prediction

Dataset	Metric		LINE	Deep-Walk	AspEm	HIN2vec	mSHINE	Graph-Sage	MAGNN	ie-HGCN	HGT	eHIN
Selected Aspect					CONCAT							
Douban Movie 2	Pre@	k=1	0.033	0.232	0.497	0.795	1.457	0.464	<u>2.318</u>	0.927	1.424	2.715
		k=10	0.063	0.175	0.831	1.030	<u>1.745</u>	0.381	1.742	1.361	1.242	2.113
		k=100	0.064	0.172	1.213	1.113	1.381	0.469	<u>1.408</u>	1.289	1.036	1.597
	Rec@	k=1	0.002	0.013	0.026	0.063	0.114	0.049	<u>0.183</u>	0.097	0.136	0.254
		k=10	0.048	0.114	0.769	0.940	<u>1.597</u>	0.330	1.544	1.233	1.118	1.962
		k=100	0.460	1.224	11.81	10.25	<u>12.36</u>	3.863	<u>12.44</u>	11.51	9.257	14.49
	MAP		0.175	0.324	1.483	1.237	1.631	0.590	<u>1.726</u>	1.503	1.221	1.970
MRR		0.417	1.058	3.978	4.570	6.601	2.270	<u>7.344</u>	5.465	5.389	8.640	
Selected Aspect					CONCAT							
CORA	Pre@	k=1	9.063	4.667	18.75	<u>22.86</u>	21.169	8.474	0.621	0.940	0.733	26.57
		k=10	3.050	1.918	4.369	<u>6.085</u>	5.572	3.112	0.269	0.836	0.280	6.770
		k=100	0.669	0.573	0.608	<u>0.917</u>	0.858	0.704	0.091	0.331	0.080	0.965
	Rec@	k=1	7.626	3.988	16.21	<u>19.86</u>	18.239	7.313	0.519	0.802	0.587	23.51
		k=10	25.42	15.66	36.22	<u>50.58</u>	46.204	25.98	2.090	6.990	2.231	56.58
		k=100	55.04	46.63	49.33	<u>74.76</u>	69.911	57.68	6.826	27.45	6.422	78.78
	MAP		13.95	8.327	23.78	<u>31.08</u>	28.44	13.86	1.127	3.133	1.224	35.70
MRR		15.75	9.247	26.02	<u>33.40</u>	30.873	15.41	1.338	3.492	1.473	37.98	
Selected Aspect					CONCAT							
Last_FM	Pre@	k=1	1.127	0.858	<u>5.472</u>	2.200	4.292	0.697	3.112	4.345	3.702	6.867
		k=10	1.068	0.317	4.088	1.980	3.374	0.762	2.473	<u>4.721</u>	4.367	5.649
		k=100	0.691	0.172	1.230	1.734	2.015	0.726	1.360	<u>2.106</u>	1.844	2.584
	Rec@	k=1	0.179	0.156	<u>0.951</u>	0.372	0.680	0.127	0.492	0.719	0.640	1.203
		k=10	1.722	0.540	6.812	3.241	5.653	1.354	4.224	<u>7.774</u>	7.467	9.482
		k=100	11.43	3.182	20.82	29.01	33.64	11.94	22.62	<u>34.68</u>	30.75	43.50
	MAP		1.236	0.524	3.766	2.938	4.295	1.226	2.866	<u>5.108</u>	4.389	6.481
MRR		4.197	1.741	12.88	7.353	12.02	3.327	9.075	<u>13.13</u>	12.38	17.54	
Selected Aspect					TMU							
IMDB-L	Pre@	k=1	0.101	0.302	7.851	1.459	<u>10.27</u>	1.862	4.076	8.203	3.520	13.04
		k=10	0.272	0.297	6.910	2.924	9.995	1.656	3.090	6.844	3.402	<u>9.587</u>
		k=100	0.179	0.418	1.674	4.694	5.808	1.534	2.450	4.639	2.744	<u>5.585</u>
	Rec@	k=1	0.002	0.045	0.497	0.062	<u>0.551</u>	0.065	0.108	0.370	0.112	0.951
		k=10	0.102	0.166	3.172	2.573	<u>4.584</u>	0.590	1.077	2.931	0.918	5.841
		k=100	0.646	2.005	6.791	22.89	<u>24.37</u>	5.292	9.232	19.56	6.719	26.32
	MAP		0.410	0.759	2.697	4.304	<u>6.056</u>	1.501	2.167	4.408	2.193	6.599
MRR		0.999	1.667	17.96	8.551	<u>23.06</u>	6.298	10.06	18.01	8.962	25.87	

and CORA, the removed edge types are set as $U-M$, $U-A$, $U-M$ and $P-A$ respectively.

Hyper-parameter settings. GraphSAGE is trained in an unsupervised learning manner as introduced in their paper. Besides, three candidate aggregators are tested here and the best result is reported. As for other GNN-based algorithms, two training methods are tested: 1) still follow GraphSAGE where the training objective is to preserve all the existing edges. 2) following MAGNN where only the target edge type (i.e. $U-M$, $U-A$, $U-M$ and $P-A$) is considered and the training objective is predicting the existence of a certain type of edges. Still, the best performance is reported. Regarding traditional algorithms, the hyper-parameters are the same as node classification except the representation learning method in AspEm where PTE is applied because of its better performance. Here, we choose to show the robustness of a node representation learning algorithm by using the same set of hyper-parameters for link prediction as node classification.

In this experiment, different objective functions proposed in different models are used to measure the respective similarity. In eHIN, the similarity between nodes in each meta-path is measured through:

$$P_j(n_c, n_t) = (\vec{h}_{n_c}^{(j)} + \vec{e}_{(T_p \rightarrow T_c)})^T \vec{h}_{n_t}^{(j)} \quad (6.25)$$

and the average of the similarity from all meta-paths will be considered as the final similarity.

Evaluation metrics. The metrics we choose to evaluate the ranking of candidate nodes include: Top-k recall, Top-k precision, Mean Average Precision(MAP) as well as Mean reciprocal rank (MRR) [99], and the details of these four metrics are given in Section 5.5.5.1.

6.4.5.2 Experimental results and discussion

The experimental result of link prediction is shown in Table 6.4 where the best performance of each dataset is highlighted in bold and the second-best performance is marked with underscore for convenient comparison.

eHIN outperforms all the other state-of-the-art algorithms from all metrics except mSHINE when the test dataset is IMDB-L while eHIN still reaches comparable performance with mSHINE on IMDB-L. The improvement of eHIN is obvious especially when compared with homogeneous-graph-based algorithms which is reasonable since the target of link prediction is to reconstruct a specific type of links while the node/edge type information is ignored in homogeneous-graph-based algorithms so that too much noise is embedded in the learned node representations to handle link prediction well. As for HIN-based methods, the performance of graph embedding methods is obviously better than that of GNN-based methods when the test dataset is CORA where the target link type is $P - A$. As introduced in Table 3.2, the node attributes for Author(A) nodes is not available in CORA. Besides, the only node type that connected with Author(A) is Paper(P) which is extremely sparse where each author node is connected with only an average of 2.6 paper nodes and this is deficient for describing author nodes. As a result, the lack of information for describing Author nodes directly limits the performance of GNN-based methods which highly rely on the message-passing from neighbor

nodes as well as the node attributes. Different from CORA, each User(U) node in IMDB-L is connected with an average of 108.4 movie nodes which helps to generate much better user node representations in GNN-based methods so that the performance of GNN-based methods is comparable with graph embedding methods when tested with IMDB-L. Similarly, in Last_FM and Douban Movie, each target node type(i.e. U , M , A) is not only connected with the other target node type but also multiple other node types, therefore, high quality node representations can still be generated through GNN-based methods with the help of neighbor nodes with other node types to reach good performance in link prediction.

6.4.6 Ablation Study

In this section, we conduct node classification experiments on different eHIN variants to evaluate the effectiveness of different components. The classification results are shown in Table 6.5 where the best performance of each dataset is highlighted in bold. Mean and Attn indicates the Mean and Attention pooling methods respectively.

It is easy to find that when label information is applied during node representation learning stage, the performance on all datasets will be improved. Especially for IMDB-S, this is because the classification performance of IMDB-S highly depends on the node attributes which have been verified in Section 6.4.4 and applying label information helps to effectively select and preserve useful node attribute information during the training stage. Even no label information is applied, the performance of eHIN on IMDB-S is still better than other graph embedding methods by integrating available node attributes information.

Theoretically, compared with Mean pooling, Attention pooling can easily reduce the noise by assigning more related neighbor nodes with higher weights and the results in Table 6.5 also verified this. Attention pooling shows an obvious improvement on Douban Movie 2(for both with and without label information situation) and CORA(without label information) and these are more complicated datasets or situations where useful information extraction is more challenge. However, for IMDB-S or CORA(with label information) which are simpler situations, the results of Attention and Mean pooling become comparable. This is because the noise in

TABLE 6.5: Classification Performance for Ablation Study in eHIN

Metric	Ratio	Douban Movie 2				IMDB-S				CORA			
Variants		w/o label info		with label info		w/o label info		with label info		w/o label info		with label info	
		Mean	Attn	Mean	Attn	Mean	Attn	Mean	Attn	Mean	Attn	Mean	Attn
Aspect		MDM	DMUMD	DMUMD	DMU-UMD	MDM	MDM	AMDMA	MDM	PPP	PPP	APPA	PPP
F1 macro	20%	16.71	16.89	17.05	17.42	51.92	52.71	60.85	60.61	67.99	68.66	72.71	73.04
	40%	15.83	16.61	16.83	17.17	55.93	55.71	62.34	62.10	69.88	70.55	74.48	74.47
	60%	15.39	15.69	16.44	16.99	56.96	56.79	62.70	63.04	70.5	71.26	74.66	74.93
	80%	15.13	15.05	15.86	16.87	56.54	56.54	62.43	63.30	70.27	71.08	75.25	74.94
F1 micro	20%	46.04	46.29	47.12	47.37	52.83	53.51	61.00	60.77	74.84	75.52	78.52	78.86
	40%	48.61	49.01	49.87	50.37	57.01	56.84	62.46	62.26	76.24	76.86	79.97	79.90
	60%	49.02	49.41	50.43	50.87	57.96	57.78	62.77	63.21	76.73	77.32	80.01	80.10
	80%	49.83	49.50	51.00	51.38	57.55	57.84	62.57	63.54	76.95	77.80	80.51	80.57

these datasets are relatively less so that the trainable transform matrix in eHIN is sufficient to handle.

6.4.7 Meta-path Information Analysis

FIGURE 6.4: Classification performance($f1$ -weighted) comparison on Douban Movie 2 between simultaneously learned and separately learned meta-path-based node representations.

As introduced in Section 6.3, the meta-path-based node representations are constructed by combining the shared node representations with the corresponding view generation functions. The shared node representations is designed to preserve the shared information among different meta-paths. To show the impact of the proposed shared node representations, we respectively conduct node classification on meta-path-based node representations which are simultaneously or separately learned(label information is ignored in this experiment for the focus on meta-path

FIGURE 6.5: Classification performance($f1$ -weighted) comparison on Cora between simultaneously learned and separately learned meta-path-based node representations.

information exploration). Simultaneously learned meta-path-based node representations indicate the representations learned through the structure introduced in Section 6.3. As for the separately learned meta-path-based node representations, only one meta-path is provided during training so that the generated node representations only contain the pure information from the corresponding meta-path.

For a simpler analysis, the classification performance in the following experiments is calculated by averaging the results from different training ratios($\{20\%, 40\%, 60\%, 80\%\}$). $F1$ -weighted score is selected as the metric in this experiment and the results are shown in Figure 6.4 and Figure 6.5⁵. For a clear display, we only show the results of symmetric meta-paths which both start and end with the key node type(i.e. M for Douban Movie and P for CORA).

In Figure 6.4 and Figure 6.5, we show the node classification performance along the epochs. The performance of node representations which are separately learned are denoted with dot lines. It is easy to find that for most of the meta-paths, simultaneously learned node representations which combine more information from other meta-paths can reach its best performance faster and this best performance is higher than the best performance which can be achieved by separately learned node representations. Specifically, in Douban Movie, simultaneously learned node representations reach the best performance at around the 300th epoch which is

⁵We only show the result of Douban Movie and Cora here since the number of meta-paths in IMDB-S is relatively small and the pattern is not obvious

0.44 while the separately learned node representations require more epochs especially for the meta-path MUM whose its best performance is around 0.42. Similar phenomenon can also be found in CORA. This verified the superiority of the proposed shared node representations of eHIN in the perspectives of effectiveness and efficiency.

However, as shown in Figure 6.5, the performance of simultaneously learned node representations shows a different trend compared with separately learned node representations. Specifically, when the meta-path-based node representations are separately learned in CORA, the classification results comparison of the meta-paths is $PPP > PTP > PAP$ which shows, as a pure meta-path, PPP can provide the most useful information for classification followed by PTP , then PAP . However, the performance comparison of simultaneously learned node representations is $PPP > PAP > PTP$ which shows a different trend.

To further explore the properties of simultaneously learned node representations, we conduct node classification on node representations which are simultaneously learned from simpler meta-path combinations. More details can be found in Figure 6.6 where we show the classification performance of simultaneously learned meta-path-based node representations which is measured with *f1-weighted* and the unit is %. The value at the coordinate (M_i, M_j) represents the *f1-weighted* score of node representations from meta-path M_i and this meta-path-based node representations are simultaneously learned with the meta-path M_j . Therefore, the value along the diagonal (M_i, M_i) indicates the classification performance of separately learned node representations based on the meta-path M_i .

In Douban Movie which is shown in Figure 6.6a, it is easy to find that, when only single meta-path is considered, MDM provides the least useful information for classification where *f1-weighted* is 16.96 and MUM provides the most useful information where *f1-weighted* reaches 42.29. However, after combining with other meta-paths, the performance of all meta-paths are improved or remain comparable to before and the changes in classification performance caused by simultaneously learning representations with other meta-paths is clearly shown in Figure 6.6b. More specifically, the performance of meta-path MAM is improved by 12.69 by combining with MUM and the performance of MDM is improved by 7.680 by combining with MUM . Combining with $MUGUM$ leads to a decrease of MUM by 1.433. In CORA, when only pure meta-paths are considered, PAP reaches the

FIGURE 6.6: (a),(c),(e) The classification performance($f1$ -weighted) of node representations learned from single meta-path and meta-path combination on Douban Movie, CORA, IMDB-S. (b),(d),(f) The changes of $f1$ -weighted in classification performance caused by simultaneously learning node representations with other meta-paths on Douban Movie, CORA, IMDB-S.

lowest score which is 48.57 while PTP reaches 59.41. However, after combining with PPP , the performance of PAP is increased by 24.61 and reaches 73.18 which is higher than the performance of PTP which is 69.07. So, combining information from other meta-paths is helpful to improve or at least maintain the original representation quality and this also explains the different trends shown in Figure 6.4 and Figure 6.5. Although IMDB-S is a relatively simple dataset, the performance improvement brought from simultaneously learning multiple meta-paths is still obvious which can also be found in Figure 6.6e and Figure 6.6f. In summary, the meta-path-based node representations which are simultaneously learned actually not only preserve the information from the current meta-path but also capture

FIGURE 6.7: Classification performance comparison between eHIN and mSHINE along epochs. (a),(b) *f1-macro* and *f1-micro* on Douban Movie 2. (c),(d) *f1-macro* and *f1-micro* on CORA.

compatible information information from other meta-paths.

6.4.8 Comparison with mSHINE

In this section, to show the higher efficiency of capturing neighbor nodes information through convolutional learning unit, we choose to compare the convergence speed of eHIN and mSHINE whose main idea is recurrently applying the sequential learning unit to each node to capture its neighbor node information. Besides, from the perspective of structure, mSHINE has the most similar learning structure as eHIN. The comparison result is shown in Figure 6.7.

In this experiment, Douban Movie and CORA are used as the experimental datasets. For a fair comparison, we choose to compare eHIN variant without label information

with mSHINE where no label information is applied either. In each figure, the classification performance is shown w.r.t training epochs which is measured as *f1-macro* and *f1-micro* score respectively. Due to the huge difference between the convergence speeds, we applied the upper x-axis and the lower x-axis to show the epochs of mSHINE and eHIN respectively. For Douban Movie, the classification performance of eHIN starts to stabilize at around the 300th epoch and reaches 0.160 and 0.490 for *f1-macro* and *f1-micro* respectively while mSHINE needs at least around 3000 epochs to achieve the comparable performance. The similar phenomena can also be observed on CORA. When the test data is CORA, eHIN starts to reach its best performance at the 200th epoch where *f1-macro* and *f1-micro* achieves around 0.701 and 0.766 respectively while mSHINE requires more than 5000 epochs where the highest score for *f1-macro* and *f1-micro* are 0.660 and 0.740 respectively which are still lower than eHIN. The fast convergence of eHIN not only shows the high efficiency of the convolution-based learning unit in spatial information capturing, but also shows the training efficiency of the meta-path-based non-sampling optimization method where all the candidate node embeddings will be simultaneously updated at each iteration. In addition to the training efficiency, the comparable or better performance achieved by eHIN further shows the effectiveness of eHIN.

6.5 Conclusion

In this chapter, a novel HIN representation learning method is proposed by fusing the techniques used in GNN-based(Section 4) and graph embedding(Section 5) methods, where, given the pre-selected meta-paths, a shared node representations matrix and a set of view generation functions are trained to generate the static meta-path-based node representations. The updating of the static node representations relies on a newly generated temporal node representations which preserves both spatial and meta-path-based sequential information through Attention pooling and GRU unit respectively. Besides, multiple optimization objectives are carefully designed to integrate the heterogeneous connectivity and label information into node representations. Through the comprehensive experiments and corresponding analysis on 5 real-world datasets, we demonstrate the capability of eHIN in modeling both the structure and node attributes information in HIN and the proposed method reaches the state-of-the-art performance in both node classification

and link prediction tasks on all datasets. The proposed shared node representations allows compatible information to be shared across different meta-paths which improves the learning efficiency and the node representations quality which is verified through our experiments. In addition, the convolution-based module cooperates with the non-sampling optimization is used to capture the spatial information, which further improved the learning efficiency.

Chapter 7

Conclusion and Future Research

7.1 Summary of completed work

Along with the widespread use of graphs in various fields, it is critical to explore effective methods for representing and analyzing these graphs in different tasks. In this thesis, we conducted a series of studies in graph representation learning which is a promising and feasible way to process graph data, and achieved good progress in graph-related tasks from different graph levels.

First, from the perspective of the GNN-based method, we focus on the graph-level representation learning and identify the problem of the low encoding efficiency when using scalar for encoding. So we proposed Capsule Graph Neural Network (CapsGNN). This novel deep learning architecture, which is inspired by CapsNet and uses node features extracted from GNN to generate high-quality graph embedding. Multiple graph representations will be generated for each graph in CapsGNN and the properties of each generated graph representation are analyzed through T-SNE. The learned graph representation quality is evaluated through the graph classification task where CapsGNN achieves state-of-the-art performance on 6 out of 10 benchmark datasets and comparable results on the rest.

Second, from the perspective of graph-embedding-based representation learning methods, we focus on node-level representation learning especially in HIN(which is a more complex situation) and proposed mSHINE. To explore suitable meta-paths when no prior knowledge is available(which is one of the main challenges for

HIN representation learning), we firstly proposed a set of criteria for meta-path selection which can well cooperate with the proposed node representation module to generate node representations with high quality. The node representation quality is evaluated through both node classification and link prediction tasks and mSHINE can achieve the state-of-the-art performance on these two tasks. Besides, the shared node presentations used in mSHINE allows node representations from different meta-paths can be learned simultaneously which improved the learning efficiency.

Third, by fusing the techniques used in GNN-based and graph-embedding-based methods, we proposed eHIN, which further improves the learning efficiency and generates node representations with better quality for downstream tasks compared with mSHINE. Unlike mSHINE where a sequential information processing unit is recurrently applied to each node to capture the topological information around the node, eHIN proposed to make use of convolutional-based unit cooperating with the attention module to capture the environment information more efficiently. Besides, by introducing non-negative sampling methods, the learning efficiency is further improved. The effectiveness of the generated node representations is evaluated through node classification and link prediction tasks. Besides, the effectiveness and properties of the shared node representations is verified through our experiments.

7.2 Future Work

In this thesis, we have identified some of the challenges in graph representation learning field and developed techniques to address these problems. However, due to the magnitude of the challenges in this area, there are still many issues and directions that are not covered in this thesis. In future, we are going to conduct the following works:

- **Application-Driven Design.** Currently, in this thesis, the applications are only done by using the publicly available real-world data. However, more specific applications need to be considered in the future, for example, the recommendation system. In this direction, algorithms need to be designed specifically for special situations instead of use general algorithms for certain application.

- **Aspects selection in HIN.** The complicated information provided by each HIN is complicated but helpful. Being able to handle this complicated information is meaningful and important. So, aspects selection in HIN is still considered as one important direction in the future. Besides, the meta-path selection criteria proposed in mSHINE is only a preliminary criteria for initial meta-path selection. A more automatic and dynamic way to select suitable meta-paths or meta-path combinations may be investigated in the future.

Author's Publications¹

Journal Articles

- **Zhang, Xinyi**, and Lihui Chen. “eHIN: An Efficient Meta-path Sharing Heterogeneous Information Network Embedding Method.” (under preparation and submission to *IEEE Transactions on Knowledge and Data Engineering* (2022))
- **Zhang, Xinyi**, Jiahao Xu, Charlie Soh, and Lihui Chen. “LA-HCN: Label-based Attention for Hierarchical Multi-label Text Classification Neural Network.” *Expert Systems with Applications* 187 (2022): 115922.
- **Zhang, Xinyi**, and Lihui Chen. “mSHINE: A Multiple-meta-paths Simultaneous Learning Framework for Heterogeneous Information Network Embedding.” *IEEE Transactions on Knowledge and Data Engineering* (2020).

Conference Proceedings

- **Xinyi, Zhang**, and Lihui Chen. “Capsule graph neural network.” In *International conference on learning representations*. 2019.

¹The superscript * indicates joint first authors

Bibliography

- [1] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and Koray kavukcuoglu. Interaction networks for learning about objects, relations and physics. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 4509–4517, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819. [1](#)
- [2] Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. Protein interface prediction using graph convolutional networks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6533–6542, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964. [1](#)
- [3] Takuo Hamaguchi, Hidekazu Oiwa, Masashi Shimbo, and Yuji Matsumoto. Knowledge transfer for out-of-knowledge-base entities : A graph neural network approach. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 1802–1808, 2017. doi: 10.24963/ijcai.2017/250. URL <https://doi.org/10.24963/ijcai.2017/250>. [1](#)
- [4] C. Shi, B. Hu, W. X. Zhao, and P. S. Yu. Heterogeneous information network embedding for recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 31(2):357–370, Feb 2019. doi: 10.1109/TKDE.2018.2833443. [1](#), [13](#), [16](#), [48](#), [60](#), [66](#), [94](#)
- [5] Kun He, Yingru Li, Sucheta Soundarajan, and John E Hopcroft. Hidden community detection in social networks. *Information Sciences*, 425:92–106, 2018. [1](#)
- [6] A. Narayanan, C. Soh, L. Chen, Y. Liu, and L. Wang. Apk2vec: Semi-supervised multi-view representation learning for profiling android applications. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 357–366, Nov 2018. doi: 10.1109/ICDM.2018.00051. [1](#)
- [7] Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. T-gcn: A temporal graph convolutional network for traffic prediction. *IEEE Transactions on Intelligent Transportation Systems*, 21(9): 3848–3858, 2020. doi: 10.1109/TITS.2019.2935152. [1](#)

- [8] Nils M Kriege, Fredrik D Johansson, and Christopher Morris. A survey on graph kernels. *arXiv preprint arXiv:1903.11835*, 2019. 1, 11
- [9] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561, 2011. 11, 12, 36
- [10] S Vichy N Vishwanathan, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11(Apr): 1201–1242, 2010. 1, 12, 36
- [11] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021. doi: 10.1109/TNNLS.2020.2978386. 2
- [12] William L. Hamilton. Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 14(3):1–159. 2
- [13] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014. 2, 13, 16, 48, 50, 53, 62, 91, 94
- [14] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016. 13, 16, 50, 53
- [15] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web, WWW '15*, pages 1067–1077, Republic and Canton of Geneva, Switzerland, 2015. International World Wide Web Conferences Steering Committee. ISBN 978-1-4503-3469-3. doi: 10.1145/2736277.2741093. 2, 14, 48, 62, 91
- [16] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008. 2, 15
- [17] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017. 15, 18, 26, 29
- [18] Hanjun Dai, Zornitsa Kozareva, Bo Dai, Alex Smola, and Le Song. Learning steady-states of iterative algorithms over graphs. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1106–1114, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL <http://proceedings.mlr.press/v80/dai18a.html>. 2, 15

- [19] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems*, pages 3856–3866, 2017. [3](#), [6](#), [25](#), [27](#), [33](#), [35](#)
- [20] Yaming Yang, Ziyu Guan, Jianxin Li, Jianbin Huang, and Wei Zhao. Interpretable and efficient heterogeneous graph convolutional network. *arXiv preprint arXiv:2005.13183*, 2020. [4](#), [16](#), [91](#)
- [21] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S. Yu, and Tianyi Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *Proc. VLDB Endow.*, 4(11):992–1003, August 2011. ISSN 2150-8097. doi: 10.14778/3402707.3402736. URL <https://doi.org/10.14778/3402707.3402736>. [4](#), [48](#), [50](#), [60](#)
- [22] Yuxiao Dong, Nitesh V. Chawla, and Ananthram Swami. Metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’17, pages 135–144, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4887-4. doi: 10.1145/3097983.3098036. [5](#), [13](#), [48](#), [50](#), [54](#), [62](#), [75](#), [76](#)
- [23] Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. Magnn: metapath aggregated graph neural network for heterogeneous graph embedding. In *Proceedings of The Web Conference 2020*, pages 2331–2341, 2020. [16](#), [18](#), [23](#), [91](#), [94](#)
- [24] Yizhou Sun and Jiawei Han. Meta-path-based search and mining in heterogeneous information networks. *Tsinghua Science and Technology*, 18(4): 329–338, 2013. doi: 10.1109/TST.2013.6574671. [4](#)
- [25] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017. [4](#), [15](#), [18](#), [62](#), [76](#), [81](#), [91](#)
- [26] Chuan Shi, Yitong Li, Jiawei Zhang, Yizhou Sun, and S Yu Philip. A survey of heterogeneous information network analysis. *IEEE Transactions on Knowledge and Data Engineering*, 29(1):17–37, 2016. [4](#), [47](#)
- [27] Yu Shi, Po-Wei Chan, Honglei Zhuang, Huan Gui, and Jiawei Han. Prep: Path-based relevance from a probabilistic perspective in heterogeneous information networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’17, page 425–434, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450348874. doi: 10.1145/3097983.3097990. URL <https://doi.org/10.1145/3097983.3097990>. [4](#)
- [28] Tao-yang Fu, Wang-Chien Lee, and Zhen Lei. Hin2vec: Explore meta-paths in heterogeneous information networks for representation learning. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge*

- Management*, pages 1797–1806. ACM, 2017. 5, 13, 14, 23, 48, 50, 63, 69, 75, 91
- [29] Geoffrey E Hinton, Alex Krizhevsky, and Sida D Wang. Transforming auto-encoders. In *International Conference on Artificial Neural Networks*, pages 44–51. Springer, 2011. 6, 27
- [30] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008. 7, 25
- [31] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990. 7, 48, 52
- [32] Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola. Kernel methods in machine learning. *The annals of statistics*, pages 1171–1220, 2008. 11
- [33] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. In *Artificial Intelligence and Statistics*, pages 488–495, 2009. 12, 36
- [34] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS’01, pages 585–591, Cambridge, MA, USA, 2001. MIT Press. URL <http://dl.acm.org.ezlibproxy1.ntu.edu.sg/citation.cfm?id=2980539.2980616>. 12
- [35] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J. Smola. Distributed large-scale natural graph factorization. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW ’13, pages 37–48, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2035-1. doi: 10.1145/2488388.2488393. URL <http://doi.acm.org.ezlibproxy1.ntu.edu.sg/10.1145/2488388.2488393>. 12
- [36] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM ’15, pages 891–900, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3794-6. doi: 10.1145/2806416.2806512. URL <http://doi.acm.org.ezlibproxy1.ntu.edu.sg/10.1145/2806416.2806512>. 12
- [37] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS’13, pages 3111–3119, USA, 2013. Curran Associates Inc. 13, 36, 51, 54, 57, 62
- [38] Frederic Morin and Yoshua Bengio. Hierarchical probabilistic neural network language model. In *Aistats*, volume 5, pages 246–252. Citeseer, 2005. 13

- [39] Shirui Pan, Jia Wu, Xingquan Zhu, Chengqi Zhang, and Yang Wang. Tri-party deep network representation. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, page 1895–1901. AAAI Press, 2016. ISBN 9781577357704. [13](#)
- [40] Cheng Li, Jiaqi Ma, Xiaoxiao Guo, and Qiaozhu Mei. Deepcas: An end-to-end predictor of information cascades. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, pages 577–586, Republic and Canton of Geneva, Switzerland, 2017. International World Wide Web Conferences Steering Committee. ISBN 978-1-4503-4913-0. doi: 10.1145/3038912.3052643. URL <https://doi-org.ezlibproxy1.ntu.edu.sg/10.1145/3038912.3052643>.
- [41] Leonardo F.R. Ribeiro, Pedro H.P. Saverese, and Daniel R. Figueiredo. Struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pages 385–394, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4887-4. doi: 10.1145/3097983.3098061. URL <http://doi.acm.org.ezlibproxy1.ntu.edu.sg/10.1145/3097983.3098061>.
- [42] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 40–48. JMLR.org, 2016. URL <http://dl.acm.org.ezlibproxy1.ntu.edu.sg/citation.cfm?id=3045390.3045396>.
- [43] Shaohua Fan, Junxiong Zhu, Xiaotian Han, Chuan Shi, Linmei Hu, Biyu Ma, and Yongliang Li. Metapath-guided Heterogeneous Graph Neural Network for Intent Recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining - KDD '19*, pages 2478–2486, Anchorage, AK, USA, 2019. ACM Press. ISBN 978-1-4503-6201-6. doi: 10.1145/3292500.3330673. URL <http://dl.acm.org/citation.cfm?doid=3292500.3330673>. [13](#)
- [44] Yu Shi, Qi Zhu, Fang Guo, Chao Zhang, and Jiawei Han. Easing embedding learning by comprehensive transcription of heterogeneous information networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2190–2199. ACM, 2018. [13](#), [48](#)
- [45] Jian Tang, Meng Qu, and Qiaozhu Mei. Pte: Predictive text embedding through large-scale heterogeneous text networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1165–1174. ACM, 2015. [14](#), [48](#), [63](#)

- [46] Shiyu Chang, Wei Han, Jiliang Tang, Guo-Jun Qi, Charu C Aggarwal, and Thomas S Huang. Heterogeneous network embedding via deep architectures. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 119–128. ACM, 2015. [14](#)
- [47] Sami Abu-El-Haija, Bryan Perozzi, and Rami Al-Rfou. Learning edge representations via low-rank asymmetric projections. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM '17*, pages 1787–1796, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4918-5. doi: 10.1145/3132847.3132959. URL <http://doi.acm.org.ezlibproxy1.ntu.edu.sg/10.1145/3132847.3132959>. [14](#)
- [48] H. Gui, J. Liu, F. Tao, M. Jiang, B. Norick, and J. Han. Large-scale embedding learning in heterogeneous event data. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 907–912, Dec 2016. doi: 10.1109/ICDM.2016.0111. [14](#)
- [49] Yu Shi, Huan Gui, Qi Zhu, Lance Kaplan, and Jiawei Han. Aspem: Embedding learning by aspects in heterogeneous information networks. In *Proceedings of the 2018 SIAM International Conference on Data Mining*. SIAM, 2018. [14](#), [22](#), [48](#), [62](#), [76](#), [91](#)
- [50] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, pages 2177–2185, 2014. [14](#)
- [51] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Y. Chang. Network representation learning with rich text information. In *Proceedings of the 24th International Conference on Artificial Intelligence, IJCAI'15*, page 2111–2117. AAAI Press, 2015. ISBN 9781577357384. [14](#)
- [52] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1365–1374. ACM, 2015. [14](#), [21](#), [36](#)
- [53] Annamalai Narayanan, Mahinthan Chandramohan, Lihui Chen, Yang Liu, and Santhoshkumar Saminathan. subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs, 2016. [14](#)
- [54] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International Conference on Machine Learning*, pages 1188–1196, 2014. [14](#), [36](#)
- [55] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. graph2vec: Learning distributed representations of graphs. *CoRR*, abs/1707.05005, 2017. URL <http://arxiv.org/abs/1707.05005>. [14](#), [36](#)

- [56] Sergey Ivanov and Evgeny Burnaev. Anonymous walk embeddings. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2186–2195, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL <http://proceedings.mlr.press/v80/ivanov18a.html>. 36, 38
- [57] Dang Nguyen, Wei Luo, Tu Dinh Nguyen, Svetha Venkatesh, and Dinh Phung. Learning graph representation via frequent subgraphs. In *Proceedings of the 2018 SIAM International Conference on Data Mining*, pages 306–314. SIAM, 2018. 14
- [58] Mohamed A Khamsi and William A Kirk. *An introduction to metric spaces and fixed point theory*, volume 53. John Wiley & Sons, 2011. 15
- [59] Shih-Yang Su, Hossein Hajimirsadeghi, and Greg Mori. Graph generation with variational recurrent neural network. *arXiv preprint arXiv:1910.01743*, 2019. 15
- [60] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. In *International Conference on Learning Representations*, 2015.
- [61] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1556–1566. Association for Computational Linguistics, 2015. doi: 10.3115/v1/P15-1150. URL <http://aclweb.org/anthology/P15-1150>.
- [62] Nanyun Peng, Hoifung Poon, Chris Quirk, Kristina Toutanova, and Wen-tau Yih. Cross-sentence n-ary relation extraction with graph lstms. *Transactions of the Association for Computational Linguistics*, 5:101–115, 2017. URL <http://aclweb.org/anthology/Q17-1008>. 15
- [63] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann Lecun. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations (ICLR2014), CBLS, April 2014*, 2014. 15
- [64] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018. 15, 81, 91
- [65] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015. 15

- [66] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of AAAI Conference on Artificial Intelligence*, 2018. 15, 16, 17, 24, 31, 34, 37, 38
- [67] Guohao Li, Matthias Müller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *The IEEE International Conference on Computer Vision (ICCV)*, 2019. 15
- [68] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=ryGs6iA5Km>. 15
- [69] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *arXiv preprint arXiv:1806.08804*, 2018. 16
- [70] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling, 2019. 16
- [71] Binbin Hu, Yuan Fang, and Chuan Shi. Adversarial Learning on Heterogeneous Information Networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining - KDD '19*, pages 120–129, Anchorage, AK, USA, 2019. ACM Press. ISBN 978-1-4503-6201-6. doi: 10.1145/3292500.3330970. URL <http://dl.acm.org/citation.cfm?doid=3292500.3330970>. 16, 17, 48, 62
- [72] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous graph transformer. In *Proceedings of The Web Conference 2020*, pages 2704–2710, 2020. 16, 18, 91
- [73] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In *The World Wide Web Conference, WWW '19*, pages 2022–2032, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-6674-8. doi: 10.1145/3308558.3313562. 16, 66, 91, 94
- [74] Xia Chen, Guoxian Yu, Jun Wang, Carlotta Domeniconi, Zhao Li, and Xiangliang Zhang. ActiveHNE: Active heterogeneous network embedding. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 2123–2129. International Joint Conferences on Artificial Intelligence Organization, 2019. doi: 10.24963/ijcai.2019/294. URL <https://doi.org/10.24963/ijcai.2019/294>. 16
- [75] James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1993–2001, 2016. 16, 17, 34

- [76] Saurabh Verma and Zhi-Li Zhang. Graph capsule convolutional neural networks. *arXiv preprint arXiv:1805.08090*, 2018. [16](#), [17](#), [24](#), [34](#), [37](#), [38](#)
- [77] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Deep neural networks for learning graph representations. In *AAAI*, pages 1145–1152, 2016. [17](#)
- [78] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 1225–1234, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939753.
- [79] Zhen Zhang, Hongxia Yang, Jiajun Bu, Sheng Zhou, Pinggang Yu, Jianwei Zhang, Martin Ester, and Can Wang. Anrl: Attributed network representation learning via deep neural networks. In *IJCAI*, volume 18, pages 3155–3161, 2018. [17](#)
- [80] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning*, pages 2014–2023, 2016. [17](#), [37](#)
- [81] Ruo-Chun Tzeng and Shan-Hung Wu. Ego-CNN: An ego network-based representation of graphs detecting critical structures, 2018. URL <https://openreview.net/forum?id=SkBHR1WRW>.
- [82] Antoine J.-P. Tixier, Giannis Nikolentzos, Polykarpos Meladianos, and Michalis Vazirgiannis. Graph classification with 2d convolutional neural networks, 2018. URL <https://openreview.net/forum?id=Hk0huyA6->.
- [83] Zhiling Luo, Ling Liu, Jianwei Yin, Ying Li, and Zhaohui Wu. Deep learning of graphs with ngram convolutional neural networks. *IEEE Transactions on Knowledge and Data Engineering*, 29(10):2125–2139, 2017. [17](#)
- [84] Heliang Zheng, Jianlong Fu, Tao Mei, and Jiebo Luo. Learning multi-attention convolutional neural network for fine-grained image recognition. In *Int. Conf. on Computer Vision*, volume 6, 2017. [17](#), [28](#)
- [85] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. [17](#)
- [86] Hongwei Wang, Jia Wang, Jialin Wang, Miao Zhao, Weinan Zhang, Fuzheng Zhang, Xing Xie, and Minyi Guo. Graphgan: Graph representation learning with generative adversarial nets. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. [17](#), [48](#)
- [87] Yizhou Sun, Yintao Yu, and Jiawei Han. Ranking-based clustering of heterogeneous information networks with star network schema. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 797–806, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-495-9. doi: 10.1145/1557019.1557107. [21](#)

- [88] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1162. URL <https://www.aclweb.org/anthology/D14-1162>. 22
- [89] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4): 19, 2016. 22
- [90] Jing Zheng, Jian Liu, Chuan Shi, Fuzhen Zhuang, Jingzhi Li, and Bin Wu. Dual similarity regularization for recommendation. In *Proceedings, Part II, of the 20th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining - Volume 9652, PAKDD 2016*, page 542–554, Berlin, Heidelberg, 2016. Springer-Verlag. ISBN 9783319317496. doi: 10.1007/978-3-319-31750-2_43. URL https://doi.org/10.1007/978-3-319-31750-2_43. 23
- [91] Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst. Matrix capsules with EM routing. In *International Conference on Learning Representations*, 2018. URL <https://openreview.net/forum?id=HJWLfGWRb>. 27
- [92] Jonas Gehring, Michael Auli, David Grangier, and Yann N Dauphin. A convolutional encoder model for neural machine translation. *arXiv preprint arXiv:1611.02344*, 2016. 28
- [93] Martin Simonovsky and Nikos Komodakis. Dynamic edgeconditioned filters in convolutional neural networks on graphs. In *Proc. CVPR*, 2017. 34, 37
- [94] Saurabh Verma and Zhi-Li Zhang. Hunt for the unique, stable, sparse and fast feature learning on graphs. In *Advances in Neural Information Processing Systems*, pages 88–98, 2017. 37
- [95] Ting Chen and Yizhou Sun. Task-guided and path-augmented heterogeneous network embedding for author identification. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 295–304, 2017. 48
- [96] Ninghao Liu, Qiaoyu Tan, Yuening Li, Hongxia Yang, Jingren Zhou, and Xia Hu. Is a Single Vector Enough?: Exploring Node Polysemy for Network Embedding. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining - KDD '19*, pages 932–940, Anchorage, AK, USA, 2019. ACM Press. ISBN 978-1-4503-6201-6. doi: 10.1145/3292500.3330967. URL <http://dl.acm.org/citation.cfm?doid=3292500.3330967>. 48, 62
- [97] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning

- Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1179. 57, 83
- [98] Léon Bottou. Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nimes*, 91(8):12, 1991. 64
- [99] P. Cui, X. Wang, J. Pei, and W. Zhu. A survey on network embedding. *IEEE Transactions on Knowledge and Data Engineering*, 31(5):833–852, 2019. 66, 70, 94, 97
- [100] Chong Chen, Min Zhang, Chenyang Wang, Weizhi Ma, Minming Li, Yiqun Liu, and Shaoping Ma. An efficient adaptive transfer neural network for social-aware recommendation. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR’19*, page 225–234, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450361729. doi: 10.1145/3331184.3331192. URL <https://doi.org.remotexs.ntu.edu.sg/10.1145/3331184.3331192>. 86, 87
- [101] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015. URL <http://arxiv.org/abs/1412.6980>. 92