Hop-Hop Relation-aware Graph Neural Networks

Li Zhang and Haiping Lu

University of Sheffield, Sheffield, UK
{lzhang72,h.lu}@sheffield.ac.uk

Abstract. Graph Neural Networks (GNNs) have been widely used in graph representation learning. However, existing popular methods can not model the relationship between central node with its different hops or types of neighbors. To address this weakness, we propose a new model, Hop-Hop Relation-aware Graph Neural Network (HHR-GNN), that introduces knowledge graph embedding method: neural tensor network (NTN) to learn the relation-scores between central node and its different hops of representation, which can provide a useful and personalized context for each node in the aggregation process. This mechanism is suitable for both homogeneous and heterogeneous graphs representation learning. Experimental results on five benchmarks show the competitive performance of our model compared to state-of-the-art GNNs, with up to 13K faster in terms of time consuming per training epoch on large heterogeneous graphs.

Keywords: GNN · NTN · Hop-Hop Relation.

1 Introduction

Graph Neural Networks (GNNs) were introduced in [11] and [22] as a generalization of recursive neural networks. In recent years, Graph Convolution Network [15] was proposed and it simplifies graph convolutions [21, 25, 6, 7, 25] by passing messages between direct (one-hop) neighbors, which is analogous to the receptive field of a convolutional kernel in Convolutional Neural Networks (CNNs). GNNs have been widely used in graph representation learning and achieved impressive performance in various application areas, such as academic citation networks [15, 28], social networks [12], knowledge graphs [23] and recommender systems [3, 34].

However, limiting each layer's messages to only one-hop neighbors seems arbitrary [1, 2], and there have been some efforts of using extended neighborhoods for aggregation in GNNs. GraphSAGE [12], Mix-Hop [1] and Lanczosnet [17] exploit multi-scale information based on the fixed-length random walk or powers of an adjacency matrix, which improve performance by utilizing the higher-order neighborhood information. Different hops of neighbors show different importance in central node's representation learning, but these methods do not consider the relationship between central node and its different-hops neighbors.

This property is even more important for heterogeneous graphs that have more complex connectivity (multiple types of nodes and edges). Different types of neighbors have significant impact on central node's representation learning. For example, a Paper's research area (class) is more determined by its Author (A type node) than Conference (C type node) generally. The majority of current GNNs assume graphs as HOmogeneous graphs (GNN-HO), and there are some attempts to apply GNNs to heterogeneous graphs (GNN-HE) [23, 30]. Relational Graph Convolutional Network (R-GCN) [23] applies a relation-specific transformation for directly linked neighbors. Another type of methods, e.g., Heterogeneous Graph Attention Network (HAN) [30] and Graph Transformer Networks (GTN) [24], are mainly based on manually designed or automatically learned meta-path, which transforms a heterogeneous graph into a homogeneous graph, then GNNs can be applied on the transformed graph [24, 30, 10] to learn the node embedding, after which the central node directly concatenates different types of neighbor's embeddings from one-hop or meta-path (higher-order). However, the manually defined or learned meta-path is a general rule that is not suitable for all nodes. In other words, the relationship between each central node and different types (or different hops) of neighbors from the meta-path are not considered in aggregation process, the similar problem as in GNN-HO.

Exploring the relationship between central node and its different hops and types of neighbors in aggregation is important for several reasons: (1) it can learn a personalized context for each node in aggregation; (2) it can greatly improve the efficiency by learning a local (within a certain-hops neighborhood) useful and personalized context, instead of manipulating the whole adjacency matrix to learn a general (global) meta-path in heterogeneous graphs as in GTN; (3) it provides insights on the effective hops or types of neighbors for prediction, which improves a model's transparency and interpretability. Currently, there are no methods to explore this.

Motivated by this, we borrow the concept from Knowledge Graph Embedding(KGE): relation-score, that captures interaction between *head entity* and *tail entity* in knowledge graph (KG) [29] and propose a universal solution applicable to both homogeneous and heterogeneous graphs' representation learning. We adopt different weight matrices for different hops or types of neighbors to get the central node and its p-hop representation. Instead of directly concatenating these embeddings, we learn the relation-scores of central node with its different-hops embeddings, which allows the central node to focus on the important hops of neighbors in the multi-scale neighborhood. To this end, we feed these embeddings to a neural tensor network (NTN) [26], a powerful method to model relational information, that can simultaneously learn different types of relationship (e.g., $r_{01}, r_{02}, ..., r_{0p}$ represent different relation-scores between central node's embedding and its one-hop, two-hop, ..., p-hop representation, respectively.) Finally, we concatenate central node's embedding and its different-hops embeddings with multiplying their corresponding relation-scores to get the central node's new embedding. In this way, the new node embedding captures both different hops of neighbors information and the relationships between central node and neighbors.

In summary, the key contributions of this paper are:

1. We propose Hop-Hop Relation-aware Graph Neural Network (HHR-GNN), a new class of GNNs for computing node embeddings that incorporate the relationship between a central node and its different hops or types neighbors. With the relation-score, HHR-GNN can learn to mix latent information from neighbors at various distances and types.
2. Our model can identify a useful and personalized meta-path for each node. Analyzing the learned relation-score benefits the interpretability and be able to provide insights on which hops or types of neighbors are more effective for predictions.
3. We prove the effectiveness of our model on both commonly used homogeneous and heterogeneous graphs for node classification tasks. Experimental results show the competitive performance of our model compared to state-of-the-art GNNs, with up to 13K faster in terms of time consuming per training epoch on large heterogeneous graphs.

2 Related Works

Our model draws inspiration from the field of GNN-HO, GNN-HE and KGE. In what follows, we provide a brief overview on related works in all fields.

Graph Neural Networks (GNNs) were introduced in [11] and [22] as a generalization of recursive neural networks that can directly deal with a more general class of graphs, e.g. cyclic, directed and undirected graphs. Most existing GNNs follow a neighborhood aggregation or "message passing" scheme. Graph convolutional network (GCN) is derived from spectral graph convolutions [21, 25, 6, 7] and simplifies the K-localized Chebynet [8, 5] by only aggregating first-order neighbors. Graph Attention Networks (GAT) [28] learns to assign different edge weights at each layer based on node features. However, they only pass messages between neighboring (one-hop) nodes in each layer, which seems arbitrary [16]. Some works generalize GCN to incorporate higher-order neighbors. GraphSAGE [12] aggregates the neighbors sampled from a fixed-length random work. MixHop and Lanczosnet [18] explore multi-scale information based on powers of adjacency matrix, essentially. Graph diffusion convolution (GDC) [16] transforms the original adjacency matrix via graph diffusion to indirecly leverage high-order neighborhood information. These models improve the performance by utilizing the higher-order neighborhood information.

For heterogeneous graph, RGCN [23] and HetGNN [35] use either distinct linear projection weight or type-specific RNN to encode features for each type of adjacent neighbors, without considering the high-order neighbors. Another type of algorithms transform a heterogeneous graph into a homogeneous graph constructed by manually meta-paths in Heterogeneous Graph Attention Network (HAN) [30] or learning a soft selection of edge types for generating useful meta-paths in Graph Transformer Networks (GTN) [24]. HAN discards all intermediate nodes along the meta-path by only considering two end nodes, which

results in information loss. GTN learns the meta-paths by many times of the multiplications of softly selected adjacency matrices, which is very time consuming, especially for large scale graphs. Instead of manipulating the whole adjacency matrix to learn the meta-path, our model utilizes the nodes' low dimensional hidden representations and knowledge graph embedding method to learn the relation-score for different types of nodes within a certain hop, which can effectively learn a personalized context for each node with multiple types of neighbors in an end-to-end fashion.

Knowledge Graph Embedding (KGE) is used to learn a scoring function of *head* and *tail* entities which evaluates an arbitrary triplet and outputs a scalar to measure the acceptability of this triplet. Based on the scoring function, KGE can be roughly categorize into two groups: translational distance models (distance-based scoring functions)and semantic matching model (similarity-based scoring function). Some widely used approaches, such as TransE [4], TransH [31], TransR [18] and TransD [13] belong to the first group, while RESCAL [19], DistMult [32] and NTN (used in our paper) [26] are the semantic matching models. Exploring different KGE models is an important direction for future work.

3 Methodology

3.1 Preliminaries

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

Definition 1. *Heterogeneous graph [33]. Heterogeneous graph is a graph with multiple types of nodes and links, each node is associated with a node type, and each link is associated with a link type. It is worth noting that the type of a link $\mathcal{E}_{ij}$ automatically defines the types of nodes v_i and v_j on its two ends.*

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

In heterogeneous graph, two nodes can be connected via different semantic paths, which are called meta-paths.

Definition 2. *Meta-path [27]. A meta-path is a path defined on the network schema in a form of $v_1 \xrightarrow{r_1} v_2 \xrightarrow{r_2} ... \xrightarrow{r_p} v_{p+1}$, where v and m are node types and link types, respectively.*

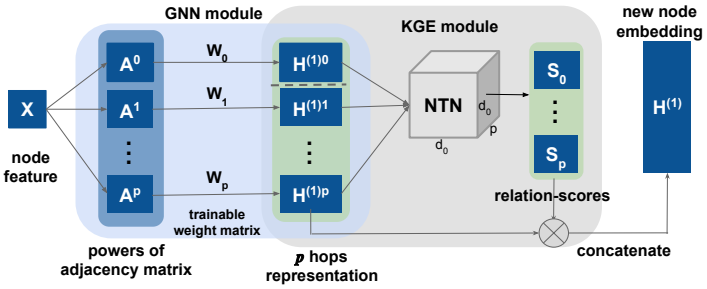


Fig. 1: HHR-GNN architecture. HHR-GNN first calculates its different hops of representation, e.g., $\mathbf{H}^{(1)_p}$ is p-hop representation that aggregates p-hops of neighbors information. Then the central node's representation and its different hops representation will be feed to a NTN model to learn the relation scores. $\mathbf{S}_p \in \mathbb{R}^{N\times 1}$ is central node with its p-hops representation's relation-scores for all nodes in a graph. Finally, we concatenate each node's embedding with its different hops of representation with multiplying its corresponding relation-score to get the new embedding.

Definition 2 defines a composite relation $R = m_1 \circ m_2 ... \circ m_p$ between node v_1 and v_{p+1}. Given the composite relation R, the adjacency matrix of the meta-path can be obtained by multiplications of adjacency matrices as:

$$\mathbf{A}_R = \mathbf{A}_{r_1}\mathbf{A}_{r_2}...\mathbf{A}_{r_p}. \tag{1}$$

For example, the meta-path Author-Paper-Conference (APC) in citation graphs, which can be represented as $A \xrightarrow{AP} P \xrightarrow{PC} C$, generates an adjacency matrix $\mathbf{A}_{APC}$ by the multiplication of $\mathbf{A}_{AP}$ and $\mathbf{A}_{PC}$. In homogeneous graph, $\mathbf{A}_{r_1} = \mathbf{A}_{r_2} ... = \mathbf{A}_{r_p}$, $\mathbf{A}_R$ means the adjacency matrix $\mathbf{A}$ multiplied by itself p times ($\mathbf{A}^p$).

3.2 Proposed Approach

In this section, we first describe the framework of our model that generalizes the concepts of GNNs with two key insights, as shown in Figure 1. First, when computing node embedding, instead of only aggregating messages computed from a node's immediate neighbors, we allow our model to aggregate message from p-hop neighborhood ($p > 1$) and use different projection matrices for different hops of neighborhood. For heterogeneous graph, HHR-GNN does not need predefined meta-paths and can learn a useful personalized context from p-hop [1] neighborhood. Secondly, before aggregating different types or hops of embedding, we learn their relation-scores in the KGE module, then concatenate different hops or types of embeddings with multiplying the corresponding relation-score. This

[1] We encourage all types of nodes to appear in the p-hop neighborhood

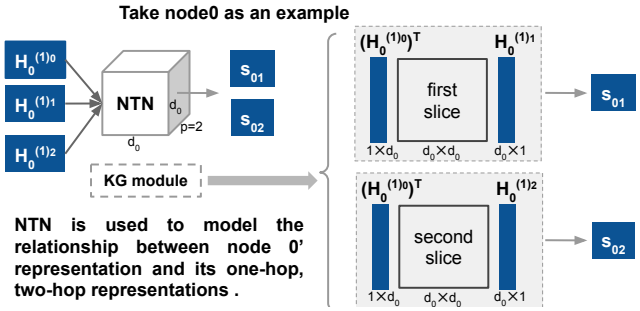


Fig. 2: Visualization of the Neural Tensor Network. Each dashed box represents one slice of the tensor layer, in this case 2 slices. Given Node 0 and its one-hop, two-hop representations learned from GNN modules, NTN utilizes two slices of tensors to model the two types of relationships: central with one-hop representation and central with two-hop representation.

allows the central node to distinguish embeddings from different types or hops in the graph.

***p*-hop Representation Learning.** Before aggregating neighborhood information, we should notice that different hops or types neighbors have different traits and their embeddings should fall in different feature space. Therefore, we design a *hop-specific* transformation matrix ($\mathbf{W}_p$) to project the node features for each hop (or type) of nodes. The *p-hop representation* can be expressed as:

$$\mathbf{H}^{(k)_p} = \sigma(\mathbf{A}^p \mathbf{H}^{(k)} \mathbf{W}_p), \tag{2}$$

where σ can be any activation function, $\mathbf{W}_p \in \mathbb{R}^{d_k \times d_{k-1}}$ is the trainable transformation matrix, $\mathbf{H}^{(k)} \in \mathbb{R}^{N \times d_k}$ is the hidden representation of the k-th layer and $\mathbf{A}^p$ is the p-th power of adjacency matrix $\mathbf{A}$. $\mathbf{H}^{(k)_p}$ is the *p-hop representation*. Note that $\mathbf{H}^{(k)_0} = \mathbf{H}^{(k)}$.

As for heterogeneous graph, *p-hop representation* can also be interpreted as a certain type of neighbors' representation. Because $\mathbf{A}^p$ can be seen as the p-th step in the meta-path, as defined in Def 2, which also automatically defines the node type (or edge type), as we defined in Def 1. For example, the adjacency matrix $\mathbf{A}_{APC}$ ($\mathbf{A}_{AP} \times \mathbf{A}_{PC}$) can be seen as two-hop (for A type nodes) or AC-type connectivity matrix. So, Eq. 2 is a general expression and can be used in both homogeneous and heterogeneous graphs to represent a certain hop or type of representation. Note, if there are different types of nodes in the same hop in heterogeneous graph, we use different transformation matrices to learn the embeddings. For example, there are two types of nodes in the same hop, and we will use $\mathbf{W}_{p1}$ and $\mathbf{W}_{p2}$ to mapping the two types of neighbors.

Relation-score Learning. Before we combine different information, a key issue is that our model should allow the central node to distinguish different hops or types of neighborhood information.

The relationship between central node's embedding and p-hop embedding can be seen as the relationship between *head entity* and *tail entity* in knowledge graph (KG). Knowledge Graph Embedding (KGE) methods aim to assign a score of how likely it is that two entities are in a certain relationship [26]. We define the relation-score as:

Definition 3. *Relation-score. Relation-score is to model the relationship between central node's embedding and a certain hop or type representation, such as* $s_{0p} = f_r(\mathbf{H}_i^{(k)}, \mathbf{H}_i^{(k)_p})$ *is the relation-score of* v_i *with its p-hop representation.*

A natural follow-up question is how to learn the relation-score. We want to model the relationship between central node with its p types of representations simultaneously and usually $p > 1$. So, we introduce an expressive neural tensor network (NTN) [26] that can relate two input entities vectors across multiple dimensions and each slice of the tensor is responsible for one type of entity pair, as shown in Fig. 2. So, after getting the embeddings, we computes the relation-score by the following NTN-based function:

$$\mathbf{S} = f(\mathbf{H}^{(k)}\mathbf{W}_R^{[1:p]}\boldsymbol{\chi}), \tag{3}$$

where f is a nonlinear activation function, $\mathbf{S} \in \mathbb{R}^{N\times p}$ is the learned relation-score for each relation type, $\mathbf{W}_R^{[1:p]} \in \mathbb{R}^{d_k\times d_k\times p}$ is a tensor and each slice of the tensor is responsible for instantiation of a relation. p types of representations $\mathbf{H}^{(k)_1}, \mathbf{H}^{(k)_2}, ..., \mathbf{H}^{(k)_p}$ can be written as a tensor $\boldsymbol{\chi} \in \mathbb{R}^{N\times d_k\times p}$ and Eq. 3 allows to learn multiple types of relationship simultaneously.

Aggregation. After learning the relation-score, we multiply different hops of representation with their corresponding relation-score, then concatenate them with central node's embedding to get the new embedding $\mathbf{H}^{(k+1)}$, as shown:

$$\mathbf{H}^{(k+1)} = \|_{i=0}^{p}\, \sigma(\mathbf{S}_i\mathbf{H}^{(k)_i}), \tag{4}$$

where $\|$ denotes column-wise concatenation, $\mathbf{S}_i$ is the relation-score with its i-th type of representation for all nodes in a graph, $\mathbf{H}^{(k)_i}$, $\mathbf{S}_0 = 1$, and $\mathbf{H}^{(k)_0} = \mathbf{H}^{(k)}$

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

$$\mathcal{L} = -\sum_{l\in\mathcal{V}_l}\sum_{f=1}^{F}\mathbf{Y}_{lf}\ln\mathbf{H}_{lf}^{(K)}\ , \tag{5}$$

Algorithm 1 The overeall process of HHR-GNN

Input: $\mathcal{G}$= $(\mathcal{V}, \mathcal{E}, \mathbf{X})$ with N nodes;
A set of adjacency matrices $\{\mathbf{A}_m\}_{m=1}^{M}$;
Feature matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$;
Labeled nodes $\mathcal{V}_l$;
Label indicator matrix $\mathbf{Y}_{lf} \in \mathbb{R}^{|\mathcal{V}_l| \times F}$;
The number of hops: p;
The number of layers K.
Output: The final embedding $\mathbf{h}_i^{(K)}$, and p-hop relation scores $\mathbf{s}_i \in \mathbb{R}^{1 \times p}$.
Calculate different powers of adjacency matrix $\mathbf{A}^1$,$\mathbf{A}^1$,...,$\mathbf{A}^p$. In homogeneous graph, $\mathbf{A}^p$ means p-hop connectivity matrix. In heterogeneous graph, $\mathbf{A}^p$ means a specific relational connectivity matrix, as shown in Eq. 1.
for k = 1,2,...,K **do**
  for each $v_i \in \mathcal{V}_l$ **do**
    Calculate different powers adjacency matrix
    Calculate its p-hop representation $\mathbf{h}_i^{(k)0}$, $\mathbf{h}_i^{(k)1}$,..., $\mathbf{h}_i^{(k)p}$
    Calculate the relation-scores: s_{i1}, s_{i2},..., s_{ip}
    Concatenate the central node and its p-hop neighbors embeddings with multiplying the corresponding relation-scores:

$$\mathbf{h}_i^{(k+1)} = \|_{m=0}^{p} \sigma(s_{im} \mathbf{h}_i^{(k)m})$$

  end for
end for
Calculate Cross-Entropy $\mathcal{L} = -\sum_{l \in \mathcal{V}_l} \sum_{f=1}^{F} \mathbf{Y}_{lf} \ln \mathbf{h}_{lf}^{(K)}$.
Back propagation and update parameters in HHR-GNN.
Return: $\mathbf{h}_i^{(K)}$ and $\mathbf{s}_i$.

where $\mathcal{V}_l$ is the set of node indices that have labels and d_K is the dimension of output features equaling to the number of classes. $\mathbf{Y}_{lf} \in \mathbb{R}^{|\mathcal{V}_l| \times F}$ is a label indicator matrix. With the guide of labeled data, we can optimize the our model via back propagation and learn the embeddings of nodes and relation-score. The overall process of HHR-GNN in shown in Algorithm 1.

Computational Complexity. Two key parts are p-hop representation (Eq. 2) learning and relation-score learning (Eq. 3). Each type or hop of neighbors share the same weight and the relation-score function is also shared by all nodes in a graph. So, the computation can be parallelized across all nodes. The computational complexity of Eq. 2 and 3 is $\mathcal{O}(p \times N \times d_k \times d_{k-1} + p \times N \times d_k \times d_k)$. As for memory requirement, it grows linearly in the size of the dataset and we perform mini-batch training to deal with this issue.

4 Experiments

We conduct semi-supervised node classification experiments on both homogeneous and heterogeneous graphs.

Table 1: Overview of the datasets.

Dataset	Nodes	Edges	Features	Classes	Train/Val./Test	Edge type
Cora	2,708	5,429	1,433	7	140/500/1,000	1
Citeseer	3,327	4,732	3,703	6	120/500/1,000	1
DBLP	18,405	67,946	334	3	800/400/2857	4 (PA, AP, PC, CP)
ACM	8,994	25,922	1,902	3	600/300/2125	4 (PA, AP, PS, SP)
IMDB	12,772	37,288	1,256	3	300/300/2339	4 (MA, AM, MD, DM)

4.1 Datasets.

We use two widely used citation homogeneous graphs: Cora and Citeseer, and three heterogeneous graphs: DBLP, ACM, and IMDB. Following [15, 24], we split the Train/Validation/Test as for the two types of graphs as shown in Table 1.

- **Cora.** The Cora dataset contains 2,708 documents (nodes) classified into 7 classes (i.e., Neural_Networks, Rule_Learning Probabilistic_Methods, ..., Reinforcement_Learning) and 5,429 citation links (edges). We treat the citation links as (undirected) edges and construct a binary, symmetric adjacency matrix. Each document has a 1,433 dimensional sparse bag-of-word feature vector and a class label.
- **Citeseer.** The Citeseer dataset contains 3,327 documents classified into 6 classes (i.e., Agents, AI, ..., ML) and 4,732 links. Each document has a 3,703 dimensional sparse bag-of-word feature vector and a class label.
- **DBLP**. DBLP is a computer science bibliography website and contains three types of nodes (papers (P), authors (A), conferences (C)), four types of edges (PA, AP, PC, CP). The authors are divided into four research areas (Database, Data Mining, Artificial Intelligence, and Information Retrieval). Each author is described by a bag-of-words representation of their paper keywords.
- **ACM**. ACM is a citation network and contains three types of nodes (papers(P), authors (A), subject (S)), four types of edges (PA, AP, PS, SP). The papers are divided into three classes (Database, Wireless Communication, Data Mining). Paper features correspond to elements of a bag-of-words represented of keywords.
- **IMDB** is an online database about movies and television and we use a subset of IMDB extracted by [30]. It contains three types of nodes (movies (M), actors (A), and directors (D)), and four types of edges (MA, AM, MD, DM). The movies are divided into three classes (Action, Comedy, Drama) according to their genre. Movie features are given as bag-of-words representations of plots.

4.2 Baselines and Experimental Setup.

For homogeneous graph, we choose four mostly related method, GCN [15], GAT [28], GraphSAGE [12]and MixHop [1]. GCN and GAT only aggregate the

first-order neighbors, and GAT enables specifying different weights to different nodes in a neighborhood (GCN can not). GraphSAGE aggregates information from a different number of hops, or search depth, away from a given node (sampled by random walk). MixHop can also leverage feature representations of neighbors from a various distances using different powers of adjacency matrix.

In our model, we model the central node with its tow-hop representation for Cora and Citeseer. Throughout experiments, we use the Adam optimizer [14] with learning rate 0.008, and set the regularization parameter to 5 $\times 10^{-4}$, the dropout is 0.6. We train all models for a maximum of 500 epochs and use early stopping with a patience of 20, i.e. we stop training if the validation loss does not decrease for 20 consecutive epochs. We use two HHR layer for Cora, and the dimension for each layer are 32 and 8, with two NTN (32$\times$ 32 $\times$ 2, 8 $\times$8 $\times$2) to learn the relationship between central and one-hop, central and two-hop respectively. For Citeseer, we use one HHR layer and set the embedding dimension to 32. We use the same architecture as in the original papers for GCN, GAT, GraphSAGE and MixHop, because these algorithms (GAT,GraphSAGE and MixHop) have many hyperparameters and the codes are downloaded from the accompany websites. We report the mean accuracy of 15 runs with random weight matrix initialization.

For heterogeneous graph, we compare with conventional network embedding methods: DeepWalk [20], metapaht2Vec [9], and GNN-based methods: GCN [15], GAT [28], HAN [30] and GTN [24], following [24]. DeepWalk, CN and GAT is originally designed for homogeneous graphs and we ignore the nodes(edges) types and perform these methods on the whole graph.

Considering the three graphs have three types of nodes, and two-hop neighborhood already contains all types of nodes in the three graphs. So, we model central node with its one-hop neighbors and two-hop neighbors relationship: AP, AC(A-P-C), AA(A-P-A); PA, PS, PP (P-A-P, P-S-P)) and MA, MD, MM (M-A-M, M-D-M) for the three datasets DBLP, ACM and IMDB respectively, based on the tasks. It should be emphasized that in heterogeneous graph, a certain hop neighbors can come from different types of nodes, we use different weight matrices to learn embeddings for different types of nodes in the same hop. For example, P has two types of neighbors in one-hop neighborhood: A and S in ACM dataset and we use two weight matrices to learn the two types of node's embeddings. We use the following sets of hyperparameters DBLP, ACM and IMDB: 0.5 (dropout rate), 5 $\times 10^{-4}$ (weight decay), two HHR layer with the hidden dimension 32. The learning rate are 0.004 (DBLP and ACM) and 0.006 (IMDB).

4.3 Experimental Results.

Node Classification Results. Based on results in Table 2 and Table 3, we conclude that our method is very competitive on both homogeneous and heterogeneous graphs.

Table 2: Node classification results for homogeneous graph. (%)

Methods	Cora	Citeseer
GCN	81.5 ± 0.42	70.3 ± 0.46
GAT	83.0 ± 0.70	72.5 ± 0.67
GraphSAGE	82.2 ± 2.70	71.4 ± 1.70
MixHop	81.9± 0.62	71.4 ± 0.81
Ours	**83.84 ± 0.77**	**72.74 ± 0.60**

Table 3: Node classification results for heterogeneous graph. (F1_score)

Methods	DBLP	ACM	IMDB
DeepWalk	63.18	67.42	32.08
metapath2vec	85.53	87.61	35.21
GCN	87.30	91.60	56.89
GAT	58.14	92.33	58.14
HAN	92.83	90.96	52.33
GTN	94.18	**92.68**	60.92
Ours	**94.65 ± 0.26**	91.83± 0.54	**61.67 ± 0.62**

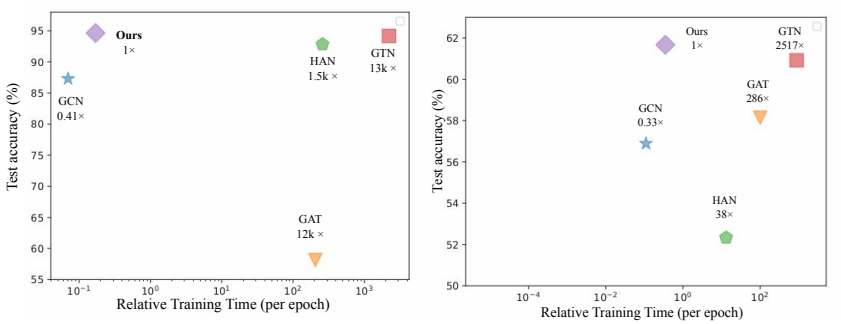
For homogeneous graph, our method leverages an enlarged (two-hop) neighborhood, which can provide more information for central node's representation learning. This is especially benefit for sparse graphs. [2]

For heterogeneous graph, our method is about 1.5% better than GTN and 9.3% better than HAN on IMDB dataset. We reason this improvement is caused by that our model can provide a personalized and relation-aware context for each node, and utilize different types of neighbors to learn the new representation for central node. HAN utilizes the manually designed meta-path to generate the homogeneous graph and other types nodes information has been lost, which may damage the central node's representation learning and make the performance unstable. The key idea of GTN is to learn a general meta-path, while this path is not suitable for every node in the graph. We encourage all types of nodes to appear within a fixed-hop neighborhood, meanwhile treat different types of nodes differently in the aggregation. Compared with HAN and GTN, our method can provide a personalized and relation-aware context for each node.

Efficiency. In Fig. 3a and Fig. 3b, we plot the performance of the state-of-the-arts GNN-HE over their training time relative to that of our model on IMDB, and DBLP datasets. The figures shows our model gets competitive performance in both accuracy and efficiently.

Fig. 3a and Fig. 3b show that GCN is the most efficient, but can not ensure the accuracy. Simply aggregating the neighbors is not suitable for heterogeneous graph that contains more complex neighborhood information than homogeneous graph. Compared with other methods, GTN has an obvious advantage in accuracy, while does not perform good in efficiency. Because GTN aims to learn an

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



(a) DBLP (b) IMDB

Fig. 3: Performance over training time on DBLP and IMDB. Ours is the highest while achieving competitive efficiency.

(a) Cora. (b) IMDB.

Fig. 4: Cora $0-1$ relation-score and $0-2$ relation-score. IMDB MA, MD and MM relation-score.

optimal meta-path by utilizing convolution to softly select adjacency matrices, which makes the final adjacency matrices are dense and the later adjacency matrices multiplication is very time consuming, especially for large graphs. For DBLP and IMDB, GTN stakes three graph transformer layers to produce the meta-paths, which needs the multiplication of the softly selected adjacency matrix: $18405\times18405\times18405$ and $12772\times12772\times12772$ for DBLP and IMDB respectively. That is why GTN spends more time than other methods. While our method learns the meta-path by utilizing the low-dimension hidden representations and a light-weight NTN model, which is very efficiency, especially for heterogeneous graphs with a few types of relations. For DBLP and IMDB, we consider three types of relations and the meta-path (relation-scores) is learned by the NTN model whose architectures are both $32\times32\times3$. For heterogeneous graphs with more types of nodes and relations, our model need more mapping matrix and more slices of NTN layers, and this will slow the computation, which we will future explore in the future works.

Interpretability. A key part in our algorithm is to treat different hops or types of neighbors differently by the learned relation-score.

We first visualize the one-hop (0-1 relation-score) and two-hop (0-1 relation-score) of the first 20 nodes on Cora, as shown in Fig. 4a. For a better comparison, we use the 100% stacked column chart and more proportions means higher relation-score in each bar. Fig. 4a shows that 0-1 relation-score is generally higher than 0-2 relation-score, which verifies that directly linked (first-order) neighbors have higher relation score than indirectly connected (second-order) neighbors.

Besides, we also show the learned relation-score of IMDB that contains four types of edges (MA, AM, MD, DM). We model the relationships of MA, MD (two types of one-hop neighbors) and MM (two-hop neighbors that formed by M-A-M and M-D-M) and visualize the three types of relation-scores in Fig 4b. Compared with Fig 4a, the relation-score is much more complexed, due to the central node is connected by different types of nodes. Fig 4b shows that each node has their preference for the different types of nodes, e.g., Node11 (Movie) has a much closed relationship with Director (the red part takes a big proportion in the bar), while Node20 has a more closed relationship with Actor.

5 Conclusion

We proposed Hop-Hop Relation-aware Graph Neural Networks, a new class of Graph Neural Networks for computing node representations that incorporates different hops and types of nodes, meanwhile their relationships with central node by the introduced NTN. We show that HHR-GNN is competitive no matter in accuracy and efficiency. Besides, it can identify the useful and personalized context or meta-path for each node, which leads to benefits in interpretability and provides insight on the effective hops or types neighbors for prediction. In our model, we just use NTN to learn the relation score, and more other KG embedding methods can be applied, which will have different influence. We leave this for future work.

References

1. Abu-El-Haija, S., Perozzi, B., Kapoor, A., Harutyunyan, H., Alipourfard, N., Lerman, K., Steeg, G.V., Galstyan, A.: Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In: ICML (2019)
2. Atwood, J., Towsley, D.: Diffusion-convolutional neural networks. In: NeurIPS (2016)
3. van den Berg, R., Kipf, T.N., Welling, M.: Graph convolutional matrix completion. arXiv preprint arXiv:1706.02263 (2017)
4. Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. In: NeurIPS. pp. 2787–2795 (2013)
5. Bronstein, M.M., Bruna, J., LeCun, Y., Szlam, A., Vandergheynst, P.: Geometric deep learning: going beyond euclidean data. IEEE Signal Processing Magazine **34**(4), 18–42 (2017)
6. Bruna, J., Zaremba, W., Szlam, A., LeCun, Y.: Spectral networks and locally connected networks on graphs. In: ICLR (2014)

7. Cho, K., van Merrienboer, B., Çaglar Gülçehre, Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using rnn encoder-decoder for statistical machine translation. In: EMNLP (2014)
8. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: NeurIPS (2016)
9. Dong, Y., Chawla, N.V., Swami, A.: metapath2vec: Scalable representation learning for heterogeneous networks. In: SIGKDD. pp. 135–144 (2017)
10. Dong, Y., Hu, Z., Wang, K., Sun, Y., Tang, J.: Heterogeneous network representation learning. In: IJCAI (2020)
11. Gori, M., Monfardini, G., Scarselli, F.: A new model for learning in graph domains. Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005. **2**, 729–734 (2005)
12. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: NeurIPS (2017)
13. Ji, G., He, S., Xu, L., Liu, K., Zhao, J.: Knowledge graph embedding via dynamic mapping matrix. In: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing. pp. 687–696 (2015)
14. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: ICLR (2015)
15. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: ICLR (2017)
16. Klicpera, J., Weißenberger, S., Günnemann, S.: Diffusion improves graph learning. In: NeurIPS (2019)
17. Liao, R., Zhao, Z., Urtasun, R., Zemel, R.: Lanczosnet: Multi-scale deep graph convolutional networks. In: ICLR (2019)
18. Lin, Y., Liu, Z., Sun, M., Liu, Y., Zhu, X.: Learning entity and relation embeddings for knowledge graph completion. In: Twenty-ninth AAAI conference on artificial intelligence (2015)
19. Nickel, M., Tresp, V., Kriegel, H.P.: A three-way model for collective learning on multi-relational data. In: ICML (2011)
20. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In: SIGKDD (2014)
21. Sandryhaila, A., Moura, J.M.: Discrete signal processing on graphs. IEEE transactions on signal processing **61**(7), 1644–1656 (2013)
22. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE Transactions on Neural Networks (2009)
23. Schlichtkrull, M., Kipf, T.N., Bloem, P., van den Berg, R., Titov, I., Welling, M.: Modeling relational data with graph convolutional networks. In: ESWC (2018)
24. Seongjun Yun, Minbyul Jeong, R.K.J.K.H.J.K.: Graph transformer networks. In: NeurIPS (2019)
25. Shuman, D.I., Narang, S.K., Frossard, P., Ortega, A., Vandergheynst, P.: The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. IEEE Signal Processing Magazine **30**, 83–98 (2013)
26. Socher, R., Chen, D., Manning, C.D., Ng, A.: Reasoning with neural tensor networks for knowledge base completion. In: NeurIPS (2013)
27. Sun, Y., Han, J.: Mining heterogeneous information networks: principles and methodologies. Synthesis Lectures on Data Mining and Knowledge Discovery **3**(2), 1–159 (2012)

28. Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph attention networks. In: ICLR (2018)
29. Wang, Q., Mao, Z., Wang, B., Guo, L.: Knowledge graph embedding: A survey of approaches and applications. IEEE Transactions on Knowledge and Data Engineering **29**(12), 2724–2743 (2017)
30. Wang, X., Ji, H., Shi, C., Wang, B., Ye, Y., Cui, P., Yu, P.S.: Heterogeneous graph attention network. In: WWW (2019)
31. Wang, Z., Zhang, J., Feng, J., Chen, Z.: Knowledge graph embedding by translating on hyperplanes. In: Twenty-Eighth AAAI conference on artificial intelligence (2014)
32. Yang, B., Yih, W.t., He, X., Gao, J., Deng, L.: Embedding entities and relations for learning and inference in knowledge bases. arXiv preprint arXiv:1412.6575 (2014)
33. Yang, C., Xiao, Y., Zhang, Y., Sun, Y., Han, J.: Heterogeneous network representation learning: Survey, benchmark, evaluation, and beyond. arXiv preprint arXiv:2004.00216 (2020)
34. Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W.L., Leskovec, J.: Graph convolutional neural networks for web-scale recommender systems. In: SIGKDD (2018)
35. Zhang, C., Song, D., Huang, C., Swami, A., Chawla, N.V.: Heterogeneous graph neural network. In: SIGKDD (2019)