A Generic Edge-Empowered Graph Convolutional Network via Node-Edge Mutual Enhancement

Pengyang Wang*
University of Central Florida
pengyang.wang@knights.ucf.edu

Jiaping Gui†
NEC Laboratories America, Inc.
jgui@nec-labs.com

Zhengzhang Chen†
NEC Laboratories America, Inc.
zchen@nec-labs.com

Junghwan Rhee
NEC Laboratories America, Inc.
rhee@nec-labs.com

Haifeng Chen
NEC Laboratories America, Inc.
haifeng@nec-labs.com

Yanjie Fu
University of Central Florida
yanjie.fu@ucf.edu

ABSTRACT

Graph Convolutional Networks (GCNs) have shown to be a powerful tool for analyzing graph-structured data. Most of previous GCN methods focus on learning a good node representation by aggregating the representations of neighboring nodes, whereas largely ignoring the edge information. Although few recent methods have been proposed to integrate edge attributes into GCNs to initialize edge embeddings, these methods do not work when edge attributes are (partially) unavailable. Can we develop a generic edge-empowered framework to exploit node-edge enhancement, regardless of the availability of edge attributes? In this paper, we propose a novel framework **EE-GCN** that achieves node-edge enhancement. In particular, the framework **EE-GCN** includes three key components: (i) Initialization: this step is to initialize the embeddings of both nodes and edges. Unlike node embedding initialization, we propose a line graph-based method to initialize the embedding of edges regardless of edge attributes. (ii) Feature space alignment: we propose a translation-based mapping method to align edge embedding with node embedding space, and the objective function is penalized by a translation loss when both spaces are not aligned. (iii) Node-edge mutually enhanced updating: node embedding is updated by aggregating embedding of neighboring nodes and associated edges, while edge embedding is updated by the embedding of associated nodes and itself. Through the above improvements, our framework provides a generic strategy for all of the spatial-based GCNs to allow edges to participate in embedding computation and exploit node-edge mutual enhancement. Finally, we present extensive experimental results to validate the improved performances of our method in terms of node classification, link prediction, and graph classification.

CCS CONCEPTS

• **Computing methodologies** → *Neural networks*; • **Information systems** → *Data mining*.

*Work done during an internship at NEC Laboratories America.
†Corresponding authors.

WWW '20, April 20–24, 2020, Taipei, Taiwan

ACM ISBN 978-1-4503-7023-3/20/04.
https://doi.org/10.1145/3366423.3380280

KEYWORDS

Graph Convolutional Networks, Node Embedding, Edge Embedding, Node-Edge Enhancement, Graph Representation Learning

ACM Reference Format:
Pengyang Wang, Jiaping Gui, Zhengzhang Chen, Junghwan Rhee, Haifeng Chen, and Yanjie Fu. 2020. A Generic Edge-Empowered Graph Convolutional Network via Node-Edge Mutual Enhancement. In *Proceedings of The Web Conference 2020 (WWW '20), April 20–24, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3366423.3380280

1 INTRODUCTION

Graph Convolutional Networks (GCNs), especially spatial-based GCNs, have been an emerging graph analysis tool and shown exceptional performances on modeling graph-structured data. Due to the high efficiency and generality, spatial-based GCNs have drawn wide attentions [1, 13, 29, 30, 38–40] in various disciplines, such as computer vision [23, 45, 46], natural language processing [2, 27, 28], and chemistry [11, 17, 35].

The idea of the spatial-based GCNs is to model node embeddings by aggregating the embeddings of neighbors [13, 30, 38]. Although promising results have been demonstrated, these classical GCNs only consider the node embeddings with node information (attributes), ignoring edge-related information. The lack of such information prevents classical GCNs from learning a holistic view of graphs, since edge information (attributes) contains semantically-rich relations among nodes, compared with plain link indicators [3, 24]. For example, in social networks, edge attributes represent different types of relationships (*e.g.*, family, alumni, colleagues) among users; in road networks, edge attributes (*e.g.*, traffic volumes, road condition) describe how passengers commute among different locations. Therefore, our hypothesis is that introducing edge information into the embedding space can further improve the performance of GCNs.

Recently, some attempts have been made to consider edge information. For example, the message passing framework is the first proposed to generate messages with both node and edge information [12]. Bettaglia *et al.* extended this message passing framework by jointly updating both node and edge representations [3]. Li *et al.* claimed that their proposed GCN-LASE is the first algorithm that incorporates edge attributes into GCNs [24]. However, these methods exhibit several limitations when integrating edge influences: (i) these methods do not work when edge attributes are (partially) unavailable; (ii) there is no mechanism to align edge embedding

with node feature space, so that edges and nodes can participate in the calculation of each other's embedding; (iii) it is unclear how node-edge mutual enhancement can be quantified in embedding calculation.

Therefore, a *generic* framework that improves current GCNs by incorporating edge information is highly desired to address the following challenges: (i) How can we develop an edge embedding initialization approach that works well even when edge attributes are (partially) unavailable? (ii) How can we devise an alignment method that can align edge embeddings with node embedding space, in order to allow edges and nodes to participate in the embedding calculation? (iii) How can we mathematically exploit node-edge mutual enhancement into embedding-update rules? Below we more formally introduce each of these challenges and how we address them in our proposed framework.

First, the initialization of embeddings is an essential step of GCNs. We aim to bring in both nodes and edges to participate in embedding learning. In a natural way, it is necessary to initialize the embeddings of both nodes and edges. Traditional methods typically initialize edge embeddings by edge attributes. Such methods have limitations, since it does not work when edge attributes are unavailable or partially available. Another possible solution is to randomly initialize edge embeddings. However, this strategy could make models sensitive to initialization and jeopardize the robustness of model performances. To address the above limitations, we propose a robust and reliable initialization strategy for edge embeddings. Specifically, we first convert the original graph into the line graph, where edges (nodes) in the original graph become nodes (edges) in the line graph. We then leverage a robust node embedding method (*e.g.*, DeepWalk) to learn the node embeddings of the line graph, which in fact are the edge embeddings of the original graph. The line graph-based initialization method can provide a more trustworthy initial embedding for edges.

Second, in order to exploit node-edge mutual enhancement, we need to introduce both edges and nodes to participate in the embedding calculation. However, node attributes (*e.g.*, surrounding area with a POI as a node) and edge attributes (*e.g.*, road condition, traffic volume with a road segment as an edge) have different semantic meanings and dimensions, and thus are not comparable directly. It is naturally desirable to align the edge embedding space with the node embedding space. Inspired by the translation-based embedding model in the knowledge graph, we propose a mapping method that maps the edge embedding into the node feature space. This is under the translation assumption that given a pair of nodes and their corresponding edge, the embedding of one node adds the embedding of the edge, resulting in the embedding of the other node. This assumption will serve as a regularization term in the loss function in order to align the edge embedding space with the node embedding space.

Third, the prior study in [3] has shown that node and edge embeddings affect each other through mutual interactions, which inspires us to combine both nodes and edges from the perspective of mutual interactions. All of the evidence shows it is promising to exploit node-edge mutual enhancement to improve GCNs. Given an edge with two end nodes, there are three types of meta interactions among them, *i.e.*, (i) the interactions from a node to the edge, (ii) the interactions from the edge to a node, and (iii) the interactions from a node to the other node. Be sure to note that the interactions from a node to the other node have already been considered by original GCNs. To take into account the three meta interactions above, we design a new updating strategy to exploit the node-edge mutual enhancement. Specifically, (i) the node embedding is updated by aggregating embeddings of neighboring nodes (original GCNs) and associated edges (via interactions from an edge to a node), and (ii) the edge embedding is updated by aggregating itself and embeddings of associated nodes (via interactions from a node to an edge). Through this updating strategy, node and edge embeddings are mutually enhanced by each other.

In summary, in this paper, we propose a generic edge-empowered framework for graph convolutional networks (**EE-GCN**). Specifically, our contributions are as follows: (1) The proposed framework **EE-GCN** is generic that can enable any current spatial-based GCNs to incorporate edge information for improving the learning performance. (2) We propose a line graph-based method for initializing edge embedding when edge attributes are (partially) unavailable. (3) We propose a translation-based mapping model that aligns node and edge embeddings into the same feature space. (4) We propose new mutually interacted updating rules to jointly model node and edge embeddings. (5) We conduct extensive experiments on real-world datasets to validate the proposed method.

2 PRELIMINARIES

2.1 Definitions

Definition 2.1. **Attributed Graphs**. In this paper, we consider undirected graphs with both node attributes and edge attributes. Formally, the attributed graphs can be represented as a graph:

$$\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{X}_V, \mathcal{X}_E), \tag{1}$$

where $\mathcal{V}$ denotes the set of nodes, $\mathcal{E}$ denotes the set of edges, $\mathcal{X}_V$ denotes the set of node attributes, and $\mathcal{X}_E$ denotes the set of edge attributes. Specifically, in some cases, edge attributes may (partially) unavailable.

Definition 2.2. **Embeddings**. In this paper, we use embeddings to denote the learned vectors for nodes and edges. Formally, let $\mathbf{h}_v$ denote the node embedding for the node v, and $\mathbf{h}_{E_{vv'}}$ denote the embedding for the edge $E_{vv'}$.

2.2 Problem Statement

In this paper, we study the problem of improving GCNs by incorporating edge information, especially when the edge attributes are (partially) unavailable. We aim to learn node and edge embeddings jointly in GCNs, which enables node embeddings to benefit from edge information. Formally, given an attributed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{X}_V, \mathcal{X}_E)$, the objective is to obtain the mapping function $f : \mathbf{x}_v \rightarrow \mathbf{h}_v, \mathbf{x}_E \rightarrow \mathbf{h}_E$. The notations used in this paper are summarized in Table 1.

3 EE-GCN: EDGE-EMPOWERED GRAPH CONVOLUTIONAL NETWORK

In this section, we introduce our framework in detail. We start with the overall framework of our proposed Edge-Empowered Graph Convolutional Network (**EE-GCN**). Then, we present the details of each component in **EE-GCN**, and summarize the algorithm.

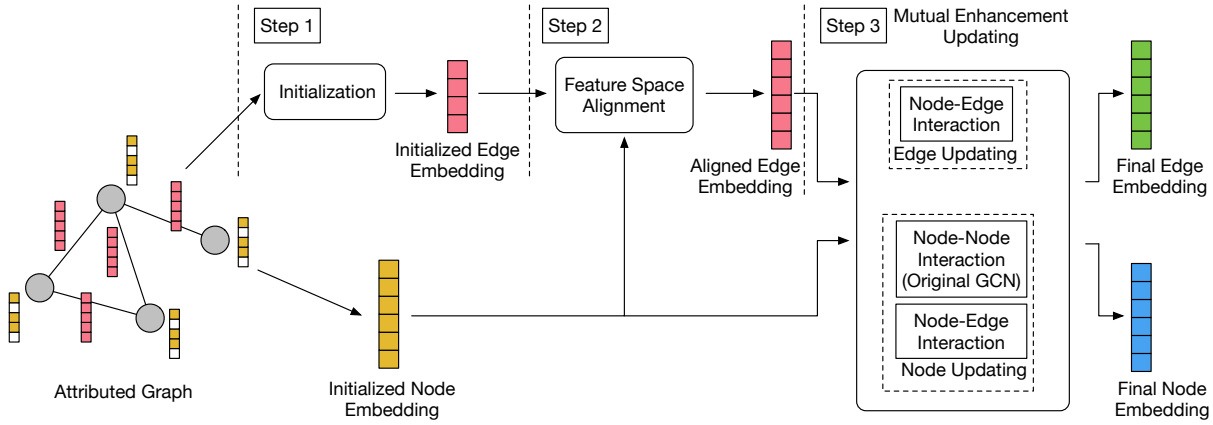


Figure 1: Overview of Edge-Empowered Graph Convolutional Network.

Table 1: Summary of Notations.

Symbol	Definition
$\mathcal{G}$	Undirected graph
$\mathcal{V}$	Node set.
$\mathcal{E}$	Edge set.
$\mathcal{X}_{(\cdot)}$	Feature set.
$\mathbf{x}_{(\cdot)}$	Attributes.
$\mathbf{h}^i_{(\cdot)}$	Embeddings at the i^{th} hidden layer.
$F_{(\cdot)}(\cdot)$	Meta-interactions.
$\mathcal{N}(v)$	Neighboring nodes of the node v.
v'	One neighboring node of the node v, $v' \in \mathcal{N}(v)$.
$E_{vv'}$	The edge between the node v and v'.
$\hat{\mathbf{h}}^i_{E_{vv'}}$	Aligned embedding (mapped from edge feature space into node feature space) of the edge $E_{vv'}$ at the i^{th} hidden layer.
$\mathbf{W}_{(\cdot)}$	Weights of the model.
$\mathbf{b}_{(\cdot)}$	Biases of the model.
$\mathcal{L}_{(\cdot)}$	Loss function.
λ	Parameter used to control the contribution of feature mapping loss.
$L(\mathcal{G})$	Line graph of the graph $\mathcal{G}$.
M	Mapping matrix for feature space alignment.
$\mathbf{z}_{(\cdot)}$	Final learned embedding.

3.1 Overview

EE-GCN aims to provide a generic framework that enables any spatial-based GCNs to incorporate edge information for improving the learning performance. An overview of **EE-GCN** is given in Figure 1. The input is an attributed graph, and the output is the node and edge embeddings. **EE-GCN** includes three key steps: (i) Initialization: the framework exploits initialization for node and edge embeddings. (ii) Feature space alignment: edge embedding is aligned into the node feature space. (iii) Node-edge mutually enhanced updating: node and edge embeddings are updated based on the interactions among nodes and edges.

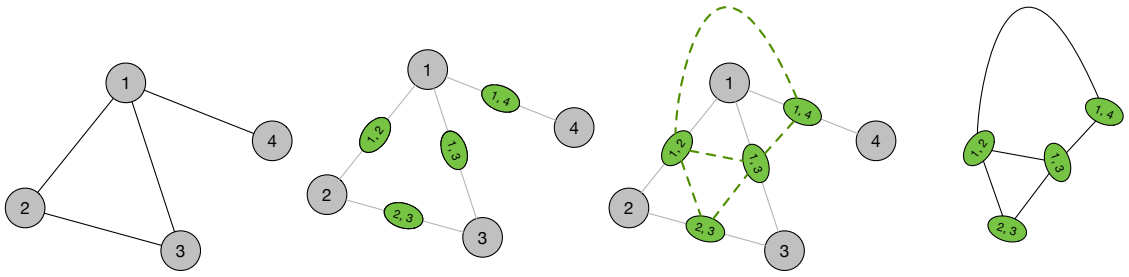
3.2 Line Graph-Based Initialization

For the first step, Line Graph-Based Initialization, our goal is to initialize both node and edge embeddings in our learning model. For the node embedding, we simply follow the initialization step of original GCNs.

However, due to the data collection difficulties, edge attributes are often (partially) unavailable under certain circumstances, such as road conditions in the road network. In such cases, the edge embedding cannot be initialized with edge attributes. An alternative solution is to initialize the edge embedding randomly. However, deep learning methods, especially embedding methods, are very sensitive to the initialization step. Random initialization would jeopardize the robustness of model performance. Therefore, we propose a line graph-based method for reliable edge embedding initialization.

The line graph was originally proposed to represent the adjacency relationship between edge in an undirected graph [14]. Specifically, given an undirected graph $\mathcal{G}$, the corresponding line graph $L(\mathcal{G})$ is a graph such that each node of $L(\mathcal{G})$ represents an edge of $\mathcal{G}$; and two nodes in $\mathcal{G}$ are adjacent if and only if their corresponding edges share a common endpoint in $\mathcal{G}$. Figure 2 shows an example of line graph construction. Take node 1, node 3, and node 4 in Figure 2a as an example. There are two edges among these three nodes: an edge $(1, 3)$ between node 1 and node 3, and an edge $(1, 4)$ between node 1 and node 4. Then, edge $(1, 3)$ and edge $(1, 4)$ will become nodes in the line graph $L(\mathcal{G})$, as shown in Figure 2b. There will also be an edge between $(1, 3)$ and $(1, 4)$ in the line graph $L(\mathcal{G})$, as shown in Figure 2c, since the two edges $(1, 3)$ and $(1, 4)$ in $\mathcal{G}$ share the same endpoint node 1.

In this work, we propose the line graph-based initialization, as shown in Line 2-6 of Algorithm 1. Specifically, given the original graph $\mathcal{G}$, we first convert it into the corresponding line graph $L(\mathcal{G})$. Then, we apply an existing node embedding method (*e.g.*, DeepWalk) to learn the node embedding from $L(\mathcal{G})$, which is essentially the edge embedding of original graph $\mathcal{G}$. Such edge embedding initialization method can be considered as a pre-trained model. Related work [14] has shown that the line graph can preserve the essential properties of edges from the original graph. Therefore, the proposed line graph-based method provides a reasonable edge embedding initialization after pre-training a node embedding model over the corresponding line graph. We validate the effectiveness of this line graph-based initialization method in Section 4.



(a) Graph $\mathcal{G}$ (b) Nodes in $L(\mathcal{G})$ constructed from edges in $\mathcal{G}$ (c) Added edges in $L(\mathcal{G})$ (d) The line graph $L(\mathcal{G})$

Figure 2: An example of line graph construction.

3.3 Node-Edge Feature Space Alignment

After the initialization, in this work, we would like to incorporate node and edge information together into modeling to improve the performance of GCNs. However, the feature size and semantic meanings for node and edge information (attributes) are not typically aligned, because the node and edge information (attributes) are used to describe different dimensions of objects. In other words, the mis-aligned node and edge information (attributes) are not comparable and cannot be combined directly.

To address this challenge, we propose a node-edge feature space alignment method, inspired by the translation-based embedding methods in knowledge graph. The translation-based embedding methods [8, 25, 42] have been proposed to align the entity and relation embedding space in knowledge graph. The main idea is to learn a mapping matrix that maps the entity embedding into the relation space, with the mapping assumption that the mapped head embedding plus the relation embedding should be equal to the tail embedding. Following the idea of translation-based embedding methods, in our problem, we can assume that given a triplet of one edge and the two endpoints $<v, E_{vv'}, v'>$ the embedding of v (v') plus the embedding of E_{AB} should be equal to that of v' (v). To achieve this, a mapping matrix M needs to be learned to map the edge embedding $\mathbf{h}_{E_{vv'}}$ into the node feature space. Formally, the mapping process is represented as:

$$\hat{\mathbf{h}}_{E_{vv'}} = M\mathbf{h}_{E_{vv'}} + b_M, \tag{2}$$

where $\mathbf{h}_{E_{vv'}}$ denotes the original edge embedding, and $\hat{\mathbf{h}}_{E_{vv'}}$ denotes the mapped edge embedding, and b_M is the bias term.

Then, the objective is to minimize the mapping loss:

$$\mathcal{L}_{\textbf{mapping}} = \sum_{v \in \mathcal{V}, v' \in \mathcal{N}(v)} |\mathbf{h}_v + \hat{\mathbf{h}}_{E_{vv'}} - \mathbf{h}_{v'}| \tag{3}$$

where $\mathcal{N}(v)$ denotes the neighbor set of node v.

Then, the aligned node and edge embedding can be fed into the updating step to learn mutually enhanced embeddings.

3.4 Node-Edge Mutually Enhanced Updating

After aligning the node-edge feature space, node and edge embeddings are still difficult to be jointly modeled, due to the complex interactions between nodes and edges. A classical GCN learns the node embedding by aggregating its neighbors' embedding, where the interactions between nodes are considered as the linear relation. However, when the edge embedding is taken into account, the interactions become more complicated. This is due to the different types of interactions that can be categorized into: (i) interactions from nodes to edges, (ii) interactions from edges to nodes, and (iii) interactions from nodes to nodes.

To address this challenge, we first define three types of meta-interactions. Then, we propose a framework to incorporate edge embedding by preserving each of the meta-interactions.

3.4.1 Meta-Interaction. The interactions among nodes and edges can be investigated from two perspectives, *i.e.* direction and quantity. For the direction side, nodes and edges interact with each other following three of directions, (i) the direction from endpoints to the associated edge, (ii) the direction from the edges to endpoints, and (iii) the direction from nodes to neighboring nodes. For the quantity side, the interaction can be regarded as how much one affects another, which can be represented as a weight in the range of $[-1, 1]$. Therefore, the quantity of interaction can be modeled by the tanh function.

Based on the above intuitions, we define three types of meta-interactions as follows. Formally, let Ws denote weights and bs denote biases,

(1) **Interactions from Nodes to Edges** $F_1(h_v, h_{E_{vv'}})$

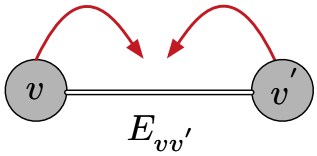


Figure 3: An example of interactions from nodes to edges.

$$F_1(h_v, h_{E_{vv'}}) = \tanh(W_1[h_v, h_{E_{vv'}}] + b_1) \tag{4}$$

(2) **Interactions from Edges to Nodes** $F_2(h_{E_{vv'}}, h_v)$

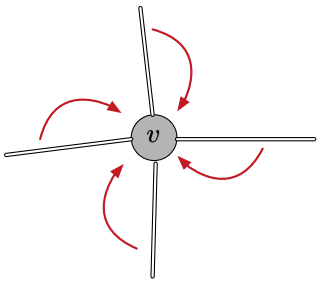


Figure 4: An example of interactions from edges to nodes.

$$F_2(h_{E_{vv'}}, h_v) = \tanh(W_2[h_{E_{vv'}}, h_v] + b_2) \tag{5}$$

(3) **Interactions from Nodes to Nodes** $F_3(h_v, h_{v'})$
The interactions from nodes to nodes have been defined by the original GCNs. For example, in GraphSAGE, the ADD aggregator defines the interactions from nodes to nodes as the summation, while the MEAN aggregator defines the interactions from nodes to nodes as the mean of embeddings. Here, we keep the definition of aggregator and denote it as $F_3(h_v, h_{v'})$.

3.4.2 Updating Scheme. Based on the above definitions of meta-interactions, we design the following updating scheme to update node and edge embeddings. Formally, let the superscript i denote the corresponding embedding in the i^{th} hidden layer (Ws and bs denote the weights and biases, respectively, the same as in Section 3.4.1), then we apply rules to update node embeddings and edge embeddings, as below:

(1) **Updating Node Embedding**
Equation 6 shows the node embedding updating rule that includes two parts: effects from neighboring nodes and effects from the associated edges. To capture the effects from neighboring nodes, we adopt the output of the original updating rules of GCNs. Since the interaction from the edges to nodes depicts how much the associated edges can affect the node, to capture the effects from associated edges, we calculate the Hadamard product between the edge embedding and the interactions from edges to nodes. Then, we sum up the effects from both neighboring nodes and associated edges as the new node embedding.

$$h_v^{l+1} = \overbrace{F_3(h_v^i, h_{v' \in \mathcal{N}(v)}^i)}^{\text{Original Updating Rules of GCNs}} + W_v \underbrace{\sum_{v' \in \mathcal{N}(v)} F_2(\hat{h}_{E_{vv'}}^i, h_v^i) \odot \hat{h}_{E_{vv'}}^i}_{\text{Effects from Edges}} \tag{6}$$

(2) **Updating Edge Embedding**
Equation 7 shows the edge embedding updating rule that includes two parts: effects from the two endpoints and effects from the edge itself. To capture the effects from the endpoints, we calculate the Hadamard product between the node embedding and the interactions from nodes to edges. Then, we sum up the effects from both nodes and the edge itself as the new edge embedding.

$$\hat{h}_{E_{vv'}}^{l+1} = \sigma(W_e \overbrace{[F_1(h_v^i, \hat{h}_{E_{vv'}}^i) \odot h_v^i, F_1(h_{v'}^i, \hat{h}_{E_{vv'}}^i) \odot h_{v'}^i,}^{\text{Effects from Two Endpoints}} \underbrace{\hat{h}_{E_{vv'}}^i}_{\text{Effects from Edge Self}}] + b_e) \tag{7}$$

3.4.3 Training Procedure. To adapt our framework into any supervised learning pipeline, such as node classification, link prediction, and graph classification, etc., we model the training procedure as an optimization problem by minimizing the cost associated with the learning. In particular, the loss function includes two parts: (1) supervised learning loss, and (2) feature space mapping loss. Formally, let $\mathcal{L}_s$ denote the supervised learning loss, the overall training loss $\mathcal{L}$ can be represented as:

$$\mathcal{L} = \lambda \mathcal{L}_s + (1 - \lambda)\mathcal{L}_{\textbf{mapping}}, \tag{8}$$

where the objective is to minimize the overall training loss $\mathcal{L}$, and λ is the weight to control the contribution of feature mapping loss to the total loss.

3.5 Algorithm Specification

Algorithm 1 shows the details about the whole **EE-GCN** framework. The framework includes three key steps: (i) initializing embedding, (ii) aligning feature space, and (iii) modeling interactions among nodes and edges. The framework takes the undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ as input, and node and edge embeddings as output. First, the framework conducts the initialization procedure (Line 1-9 in Algorithm 1). For the procedure of node embedding initialization, we follow the initialization steps of the classical GCNs. For the procedure of edge embedding initialization, when the edge attributes are available, we initialize the edge embedding with edge attributes; otherwise, we convert the original graph into its line graph, and then apply Deepwalk to learn the node embedding of line graph (*i.e.*, the edge embedding of the original graph). Then, we align the feature space by learning the mapping matrix that maps node embedding into the edge feature space (Line 10-13 in Algorithm 1). In addition, we jointly update the node and edge embedding by incorporating three types of meta-interactions (Line 15-24 in Algorithm 1). The output of the final layer will be learned embeddings of nodes and edges Finally, we put the framework in a supervised learning pipeline (*e.g.*, node classification, link prediction, and graph classification, etc) and jointly minimize the supervised learning loss and feature alignment loss (shown as in Equation 8).

4 EXPERIMENTS

This section details the empirical evaluation of our proposed framework in terms of various tasks: node classification, link prediction, and graph classification.

4.1 Data Description

We evaluate our framework **EE-GCN** on four datasets with edge attributes, *i.e.*, *BZR_MD* [19, 37], *DFHR_MD* [19, 37], *ER_MD* [19, 37], and *FIRSTMM_DB* [31]. The statistics of the datasets are shown in Table 2.

(1) *BZR_MD*: BZR is a set of 405 ligands for the benzodiazepine receptor. BZR_MD is the reduced version with $T_c = 0.95$ (Tanimoto coefficient) threshold of 2-D (structural) fingerprints [37].
(2) *DFHR_MD*: DFHR is a set of 756 inhibitors of dihydrofolate reductase. DFHR_MD is the reduced version with $T_c = 0.90$ (Tanimoto coefficient) threshold of 2-D (structural) fingerprints [37].
(3) *ER_MD*: ER is a set of 1009 estrogen receptor ligands. ER_MD is the reduced version with $T_c = 0.85$ (Tanimoto coefficient) threshold of 2-D (structural) fingerprints [37].

Algorithm 1: EE-GCN: Edge-Empowered Graph Convolutional Network.

Input : Undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$;
Input node features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$;
Edge attributes $\mathbf{x}_{E_{vv'}}$;
Number of hidden layers i;
Neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$;
Non-linearity σ;
Mapping matrix M;
Bias matrix b_M ;
Weight matrix for node embedding W_v^i, $\forall i \in \{1, \cdots, I\}$;
Weight matrix for edge embedding $W_{E_{vv'}}^i$, $\forall i \in \{1, \cdots, I\}$;
Bias vector for node embedding b_v;
Bias vector for edge embedding b_e;
DeepWalk function DEEPWALK;
Updating rule of node embedding from original GCNs f;

Output: Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$, vector representations $\mathbf{z}_{E_{vv'}}$ for all $E_{vv'} \in \mathcal{E}$

```
// Initialization for Node Embedding.
h_v^0 ← x_v, ∀v ∈ V;
// Intitialization for Edge Embedding.
if IsAvailable(x_{E_vv'}), ∀E_vv' ∈ E then
    h^0_{E_vv'} ← x_{E_vv'}, ∀E_vv' ∈ E;
else
    // Line Graph-Based Initialization.
    Convert G into line graph L(G);
    h^0_{E_vv'} ← DEEPWALK(L(G)), ∀E_vv' ∈ E;
// Feature Space Alignment.
for v ∈ V do
    v' ← N(v);
    ĥ^0_{E_vv'} ← M × h^0_{E_vv'} + b_M
// Node-Edge Mutually Enhanced Updating.
for i = 1, ··· , I do
    for v ∈ V do
        v' ← N(v);
        // Updating Node
        h^i_v ← f(h^{i-1}_v, h^{i-1}_{v'}) + W_v Σ_{v'} F_2(ĥ^i_{E_vv'}, V^i_A) ⊙ ĥ^{i-1}_{E_vv'};
        // Updating Edge
        ĥ^i_{E_vv'} ← σ(W_e[F_1(h^{i-1}_v, ĥ^{i-1}_{E_vv'}) ⊙ h^i_v,
                    F_1(h^{i-1}_{v'}, ĥ^{i-1}_{E_vv'}) ⊙ h^{i-1}_{v'}, ĥ^{i-1}_{E_vv'}] + b_e);
z_v ← h^I_v, ∀v ∈ V;
z_{E_vv'} ← h^I_{E_vv'}, ∀E_vv' ∈ E;
```

(4) *FIRSTMM_DB*: FIRSTMM is a set of 41 simulated 3D point clouds of various household objects for (semantic and graph-based) object category prediction. FIRSTMM_DB is obtained from a previously defined 3D mesh of the object by up-sampling points using midpoint surface subdivision [31].

All of these four datasets can be downloaded online[1]. Note that since *BZR_MD*, *DFHR_MD* and *ER_MD* are fully-connected graphs, we only use *FIRSTMM_DB* for the link prediction task. We use *BZR_MD*, *DFHR_MD*, and *ER_MD* for the tasks of node classification and graph classification. To generate the initialized node embedding of these datasets, we leverage DeepWalk with $dim = 100$ as the node attributes.

4.2 Experimental Settings

To evaluate the proposed framework, in the high level, we conduct experiments on two types of GCNs: no-edge-empowered (original) and edge-empowered (**EE-GCN**). For the edge-powered type, we have two different initialization versions. Then, we compare the performance of GCNs in different tasks in terms of two types of aggregators *ADD* and *MEAN* [13]. We describe the experimental settings below.

4.2.1 Comparison Methods. For comparison, we evaluate three current spatial-based GCNs and their edge-empowered versions based on our proposed framework.

(1) GraphSAGE: an inductive GCN that learns node representations by sampling and aggregating local neighbors' representations [13].
(2) K-GNN: a GCN variant with higher-order graph structures at multiple scales [30].
(3) GAT: a Graph Attention Network that adopts attention mechanisms to aggregate neighbors' representations by learning the relative weights between two connected nodes [38].
(4) *-Edge: the edge-empowered GCN version (GraphSAGE, GraphSAGE, K-GNN, or GAT) with the original edge attributes as the initialization of edge representations.
(5) *-LineEdge: the edge-empowered GCN version (GraphSAGE, GraphSAGE, K-GNN, or GAT) by initializing edge representations based on the line graph.
(6) *-RandomEdge: the GCN version (GraphSAGE, GraphSAGE, K-GNN, or GAT) with random edge representation initialization.

4.2.2 Node Classification. In the node classification task, for each of the graphs in each dataset, we randomly select 50% nodes from each class as the training set, 25% nodes as the validation set, and the remaining 25% nodes as the testing set. Note that in some graphs of *BZR_MD*, *DFHR_MD* and *ER_MD*, the number of nodes in some classes is less than three. These classes can not be split into the training, validation and testing set. Hence, we remove these graphs in our experiment. In the end, we have 276 graphs of *BZR_MD*, 389 graphs of *DHFR_MD*, and 302 graphs of *ER_MD* for evaluation.

To conduct the experiment, we adopt the implementation of node classification framework[2]. We set the learning rate to 0.05, the dropout for node representations to 0.8, and the number of

[1] https://ls11-www.cs.tu-dortmund.de/staff/morris/graphkerneldatasets
[2] https://github.com/rusty1s/pytorch_geometric/tree/master/benchmark/citation

Table 2: Statistics of the experimental datasets.

Name	Num. of Graphs	Num. of Graph Classes	Num. of Node Classes	Avg. Num. of Nodes	Avg. Num. of Edges	Node Attribute Dimensions	Edge Attribute Dimensions
BZR_MD	306	2	8	21.30	225.06	-	1
DHFR_MD	393	2	7	23.87	283.01	-	1
ER_MD	446	2	10	21.33	234.85	-	1
FIRSTMM_DB	41	11	5	137.27	3074.10	1	2

Table 3: Average accuracy of node classification w.r.t. MEAN and ADD aggregators.

	BZR_MD (MEAN)	DFHR_MD (MEAN)	ER_MD (MEAN)	BZR_MD (ADD)	DFHR_MD (ADD)	ER_MD (ADD)
GraphSAGE	0.810 ± 0.135	0.701 ± 0.195	0.895 ± 0.022	0.571 ± 0.411	0.542 ± 0.321	0.686 ± 0.291
GraphSAGE-Edge	**0.833 ± 0.146**	**0.714 ± 0.229**	**0.899 ± 0.013**	**0.662 ± 0.306**	0.556 ± 0.315	**0.732 ± 0.242**
GraphSAGE-LineEdge	0.799 ± 0.083	0.666 ± 0.190	0.861 ± 0.067	0.659 ± 0.262	**0.561 ± 0.300**	0.730 ± 0.249
GraphSAGE-RandomEdge	0.792 ± 0.129	0.621 ± 0.226	0.799 ± 0.115	0.582 ± 0.253	0.443 ± 0.212	0.726 ± 0.272
K-GNN	0.722 ± 0.235	**0.657 ± 0.243**	0.855 ± 0.126	0.494 ± 0.421	0.541 ± 0.327	0.606 ± 0.302
K-GNN-Edge	**0.776 ± 0.133**	0.650 ± 0.212	**0.860 ± 0.112**	**0.665 ± 0.263**	0.615 ± 0.304	**0.732 ± 0.221**
K-GNN-LineEdge	0.771 ± 0.226	0.647 ± 0.208	0.857 ± 0.129	0.658 ± 0.316	**0.616 ± 0.269**	0.727 ± 0.244
K-GNN-RandomEdge	0.741 ± 0.197	0.655 ± 0.336	0.845 ± 0.117	0.661 ± 0.275	0.593 ± 0.185	0.689 ± 0.306
GAT	0.310 ± 0.387	0.423 ± 0.329	0.465 ± 0.426	0.326 ± 0.395	0.318 ± 0.312	0.428 ± 0.396
GAT-Edge	**0.737 ± 0.187**	**0.641 ± 0.270**	**0.833 ± 0.118**	**0.428 ± 0.233**	0.455 ± 0.273	**0.568 ± 0.334**
GAT-LineEdge	0.733 ± 0.214	0.640 ± 0.191	0.829 ± 0.143	0.414 ± 0.371	**0.459 ± 0.280**	0.563 ± 0.361
GAT-RandomEdge	0.702 ± 0.269	0.611 ± 0.191	0.833 ± 0.147	0.398 ± 0.292	0.426 ± 0.364	0.501 ± 0.224

Table 4: Average AUC of link prediction over FIRSTMM_DB w.r.t. MEAN and ADD aggregators.

	MEAN	ADD
GraphSAGE	0.971 ± 0.007	0.976 ± 0.006
GraphSAGE-Edge	**0.981 ± 0.005**	**0.980 ± 0.006**
GraphSAGE-LineEdge	0.978 ± 0.006	0.980 ± 0.007
GraphSAGE-RandomEdge	0.970 ± 0.007	0.977 ± 0.006
K-GNN	0.970 ± 0.001	0.973 ± 0.006
K-GNN-Edge	**0.989 ± 0.006**	0.977 ± 0.009
K-GNN-LineEdge	0.977 ± 0.007	**0.979 ± 0.007**
K-GNN-RandomEdge	0.969 ± 0.009	0.979 ± 0.013
GAT	0.977 ± 0.008	0.974 ± 0.010
GAT-Edge	**0.985 ± 0.005**	**0.978 ± 0.008**
GAT-LineEdge	0.983 ± 0.006	0.972 ± 0.005
GAT-LineEdge	0.978 ± 0.011	0.966 ± 0.005
GAT-RandomEdge	0.969 ± 0.020	0.961 ± 0.017

hidden layers to 2. In addition, for edge-empowered versions, we set the dropout for edge representations to 0.8, and $\lambda = 0.9$.

We evaluate the performance of different GCNs on the task of node classification in terms of average accuracy. To this end, we first calculate the classification accuracy for each graph in each dataset. Then, we calculate the average accuracy over all graphs in each dataset as the final evaluation metric.

4.2.3 Link Prediction. In the task of link prediction, since *BZR_MD*, *DFHR_MD*, and *ER_MD* are fully-connected graphs, we only conduct experiment on *FIRSTMM_DB* to evaluate the model performance. We adopt the SEAL framework [48] for the implementation of link prediction. For each of the graphs in each dataset, we first sample the positive and negative links. Then, among both positive and negative links, we randomly select 90% for training and remaining 10% for testing. In the experiment, we set the learning rate to 0.05. For edge-empowered versions, we set $\lambda = 0.9$.

We evaluate the performance of different GCNs on the task of link prediction in terms of average AUC. To this end, we first calculate the AUC for each of the graphs in each dataset. Then, we calculate the average AUC value over all graphs in each dataset as the final evaluation metric.

4.2.4 Graph Classification. We evaluate all four datasets in the task of graph classification. To do the experiment, we adopt the implementation of graph classification framework[3]. We set the number of hidden layers = 3, the hidden size = 128, and the batch size = 10. In addition, for edge-empowered versions, we set $\lambda = 0.9$.

We evaluate the performance of different GCNs on the task of graph classification in terms of average accuracy. To this end, we first run the experiment with 4-fold cross validation. Then, we calculate the average accuracy over all folds. All experiments were conducted on Ubuntu 18.04.3 LTS, Intel(R) Core(TM) i9-9920X CPU @ 3.50GHz, with Titan RTX and memory size 128G.

4.3 Experimental Results

4.3.1 Node Classification. In the node classification task, we compare the original GCN models with two edge-empowered versions. Table 3 shows the average accuracy of different GCNs.

Overall, in the high level, **EE-GCN** (*i.e.*, *-Edge, *-LineEdge) outperforms the original GCNs (no-edge-empowered), except for the K-NN group on the *DFHR_MD* dataset with MEAN aggregator. In this case, even thougn the average accuracy of K-NN is slightly higher than K-NN-Edge and K-NN-LineEdge, the variance of K-NN is larger than the edge-empowered versions. This indicates the

[3]https://github.com/rusty1s/pytorch_geometric/tree/master/benchmark/kernel

Table 5: Average accuracy of graph classification w.r.t. ADD and MEAN aggregators.

	BZR_MD (MEAN)	DFHR_MD (MEAN)	ER_MD (MEAN)	BZR_MD (ADD)	DFHR_MD (ADD)	ER_MD (ADD)
GraphSAGE	0.476 ± 0.042	0.679 ± 0.003	0.585 ± 0.017	0.477 ± 0.066	0.547 ± 0.042	0.493 ± 0.030
GraphSAGE-Edge	**0.699 ± 0.032**	**0.753 ± 0.114**	**0.685 ± 0.042**	**0.517 ± 0.059**	**0.580 ± 0.147**	**0.522 ± 0.009**
GraphSAGE-LineEdge	0.697 ± 0.198	0.749 ± 0.139	0.682 ± 0.175	**0.517 ± 0.059**	0.577 ± 0.075	0.520 ± 0.026
GraphSAGE-RandomEdge	0.683 ± 0.207	0.724 ± 0.139	0.663 ± 0.219	0.503 ± 0.198	0.578 ± 0.142	0.500 ± 0.143
K-GNN	0.477 ± 0.072	0.679 ± 0.003	0.598 ± 0.009	0.493 ± 0.031	0.570 ± 0.059	0.497 ± 0.053
K-GNN-Edge	**0.751 ± 0.221**	0.825 ± 0.093	**0.689 ± 0.113**	**0.508 ± 0.023**	0.596 ± 0.047	**0.504 ± 0.056**
K-GNN-LineEdge	0.740 ± 0.257	**0.828 ± 0.018**	0.686 ± 0.184	0.507 ± 0.025	**0.598 ± 0.052**	0.503 ± 0.047
K-GNN-RandomEdge	0.727 ± 0.229	0.803 ± 0.191	0.661 ± 0.263	0.507 ± 0.039	0.572 ± 0.039	0.503 ± 0.099
GAT	0.513 ± 0.000	0.679 ± 0.003	0.594 ± 0.001	**0.503 ± 0.019**	0.679 ± 0.003	0.594 ± 0.001
GAT-Edge	**0.548 ± 0.036**	**0.683 ± 0.040**	**0.618 ± 0.047**	0.495 ± 0.053	**0.681 ± 0.023**	**0.596 ± 0.004**
GAT-LineEdge	0.546 ± 0.050	0.679 ± 0.003	0.594 ± 0.001	0.493 ± 0.025	0.679 ± 0.029	0.594 ± 0.001
GAT-RandomEdge	0.533 ± 0.097	0.680 ± 0.012	0.595 ± 0.011	0.490 ± 0.047	0.681 ± 0.057	0.590 ± 0.014

node classification can be improved with the contribution of edge information. In other words, the node classification of spatial-based GCNs effectively benefit from our edge-empowered framework via node-edge mutual enhancement.

Comparing two different versions of edge initialization in **EE-GCN**, we can observe that the results of edge attributes-based initialization are better than (but close to) those of line graph-based version. In the case of *DFHR_MD* (ADD), the performance of line graph-based initialization is even slightly better. The potential explanation is that line graph preserves the essential topology information of edges in the original graph. DeepWalk over the line graph can be considered as a pre-train styled initialization that takes advantage of topology information effectively.

In terms of aggregators, we can see that the MEAN aggregator helps GCNs achieve a better performance than the ADD aggregator in the task of node classification. This observation is similar to the results in [13].

4.3.2 Link Prediction. We evaluate the performance of link prediction task over the *FIRSTMM_DB* dataset whose results are shown in Table 4. From the table, we can observe that **EE-GCN** outperforms the original GCNs. The traditional link prediction in original GCNs aims to determine whether there exists an edge between nodes only based on node embeddings. Incorporating edge information contributes to the learning of node embeddings.

For the two different versions of **EE-GCN**, edge attributes-based initialization typically has a slightly better performance than line graph-based, which is similar to the observation in the node classification task (Section 4.3.1). In terms of aggregators, the MEAN aggregator is overall slightly better than the ADD, but the difference is trivial.

4.3.3 Graph Classification. Table 5 shows the results of graph classification task. Overall, we have similar observations as in the node classification task. **EE-GCN** outperforms the original GCNs on the graph classification task. In addition, for the two different versions of **EE-GCN**, the results of edge attributes-based initialization are better than (but close to) those of line graph-based. In terms of aggregators, the results of MEAN aggregator are much better than those of ADD.

4.4 Analysis of λ

We study the effects of the hyperparameter λ over the performance of **EE-GCN**. In the training procedure, the feature mapping loss is to regularize the framework to align the node and edge feature space. λ is the parameter to control the contribution of feature mapping loss to the total loss. The larger the λ is, the less the contribution of feature mapping loss is.

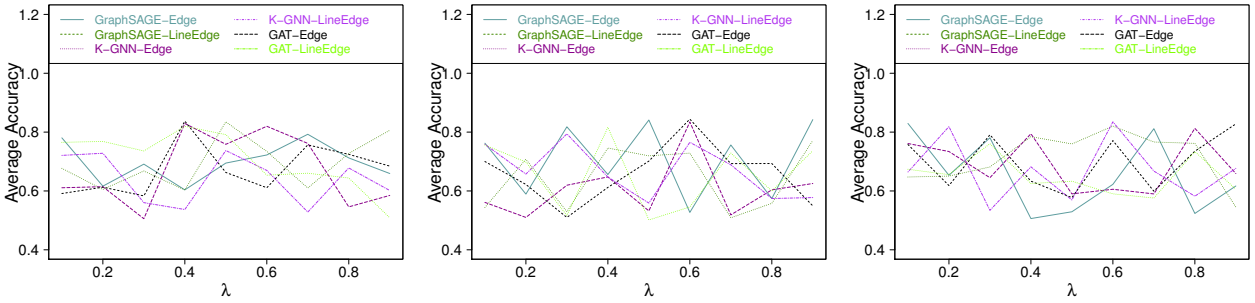
To study the effects of λ, we set the scale of λ into $[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]$, while keeping other hyperparameters the same. We run the experiment across all the values of λ for all three tasks of node classification, link prediction and graph classification. Then, we plot the results into Figure 5, Figure 6 and Figure 7. Due to the page limitation, we only illustrate the results with the MEAN aggregator.

Figure 5, Figure 6, and Figure 7 show that the model performance is quite sensitive to the value of λ. For the node classification task on the same dataset, all the models achieve the best performance in the middle range of λ, while for the graph classification task, there is no such pattern. However, for the link prediction task, the edge-empowered framework can roughly achieve the best performance in the upper and lower range of the λ values. Above results indicate that different tasks require different levels of feature space alignment.

4.5 Analysis of Training Time

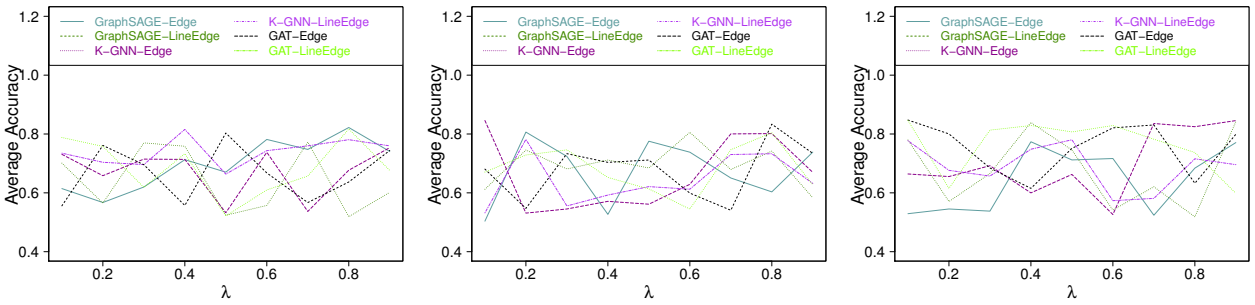
We study the training time for both the original GCNs and edge-empowered versions (*i.e.* *-Edge and *-LineEdge). We run the same parameter setting for 100 times. Then we calculate the average training time per epoch. We test the training time for node classification, link prediction and graph classification with the MEAN and ADD aggregator. Due to the page limitation, we only illustrate the results on the MEAN aggregator.

Figure 8, Figure 9 and Figure 10 show the results. We can observe that in the task of node classification and link prediction, the training time of edge-empowered versions is 1.3 times longer than the original GCNs. But for the graph classification, the difference between edge-empowered versions and the original GCNs is relatively small. Because of edge embedding, **EE-GCN** has more parameters to update and additional computation about the edge



(a) Node classification on *BZR_MD*. **(b) Node classification on *DFHR_MD*.** **(c) Node classification on *ER_MD*.**

Figure 5: Average accuracy of node classification w.r.t. different λ values.



(a) Graph classification on *BZR_MD*. **(b) Graph classification on *DFHR_MD*.** **(c) Graph classification on *ER_MD*.**

Figure 6: Average accuracy of graph classification w.r.t. different λ values.

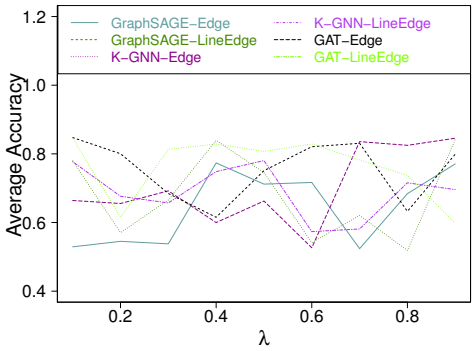


Figure 7: Average AUC of link prediction w.r.t. different λ values.

embedding, which leads to the relative higher time consuming. However, since it only needs very small numbers of training epoch for edge-empowered versions to outperform the original GCNs, the trade-off of relative higher time consuming is acceptable in practice.

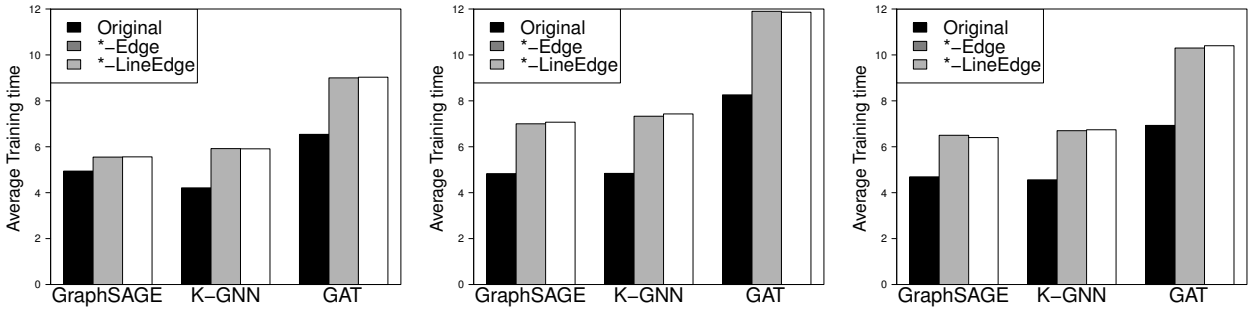
5 RELATED WORK

Graph Convolutional Networks. Our work is connected with Graph Convolutional Networks (GCNs). GCNs generalize the convolution operation from grid-structured data to graph-structured data. The core idea of GCNs is to learn node representations by aggregating neighbors' representations [44]. GCNs can be categorized into two different types: (i) spectral-based GCNs and (ii) spatial-based GCNs.

The spectral-based GCNs design the convolution operation over graph by introducing filters from the perspective of graph signal processing [44]. The graph convolutions are regraded as removing graph signal noises [36]. The spectral-based GCNs assume the filter as a set of learnable parameters and consider graph signals with multiple channels [10, 18]. Some solutions are further proposed to make improvements over GCNs using other symmetric matrices, like Adaptive Graph Convolutional Network (AGCN) [22] and Dual Graph Convolutional Network (DGCN) [49].

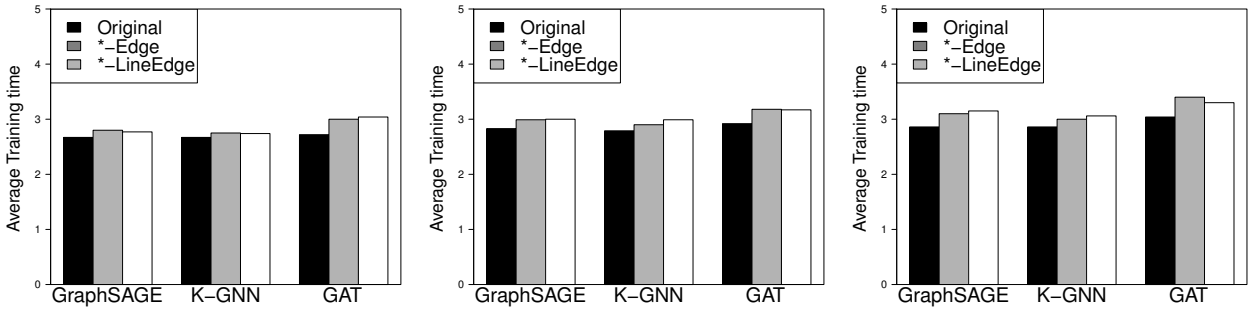
The spatial-based GCNs define the graph convolutions by explicitly aggregating local information based on the topology structure. For example, Neural Network for Graphs (NN4G) is proposed to sum up a node's neighborhood information directly by applying residual connections and skip connections to memorize information over each layer [29]. Diffusion Convolutional Neural Network (DCNN) assumes graph convolutions as a diffusion process by assigning a certain transition probability for information transferred from one node to another [1]. GraphSAGE is proposed to aggregate neighbor's information through a sampling strategy [13]. Graph Attention Network (GAT) adopts the attention mechanism to learn the relative weight to quantify the contributions of neighboring nodes [38]. Morris et al. propose k-GNNs by taking higher-order graph structures into account at multiple scales [30].

These classical GCNs only consider aggregating neighboring node information, but ignoring edge information. While some



(a) Node classification on *BZR_MD*. **(b) Node classification on *DFHR_MD*.** **(c) Node classification on *ER_MD*.**

Figure 8: Average training time per epoch in node classification.



(a) Graph classification on *BZR_MD*. **(b) Graph classification on *DFHR_MD*.** **(c) Graph classification on *ER_MD*.**

Figure 9: Average training time per epoch in graph classification.

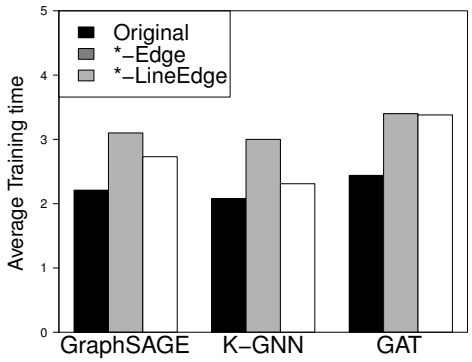


Figure 10: Average training time per epoch in link prediction.

work [3, 12, 24] attempts to incorporate edge information into GCNs, the common drawback is that they are specifically defined and limited, which prevents them from being applicable to current well-performed GCNs. To address this drawback, in this paper, we propose a flexible edge-empowered GCN framework that enables current spatial-based GCNs to benefit from the edge information.

Network Embedding. Our work is also connected with representation learning that can be categorized into three main approaches: (i) the probabilistic models, (ii) the geometrically motivated manifold-learning approaches, and (iii) the reconstruction-based algorithms related to auto-encoder.

The key idea of the probabilistic model-based approaches is to use unsupervised feature learning to learn a hierarchy of features one level at a time [4, 7, 15, 21, 32, 34, 41]. In the second category, the large majority of the algorithms adopt a non-parametric approach, based on a training set nearest neighbor graph [9, 26, 33, 43, 43]. Compared to probabilistic models, the auto-encoder based methods do not need complicated posterior distributions thanks to the use of latent variables. Auto-encoders can directly parameterize features or representation functions, and learn a direct encoding [5, 6, 16, 20, 47, 50].

6 CONCLUSIONS

Graph Convolutional Networks (GCNs) have shown to be a powerful tool for modeling graph-structured data. Recently, few methods have been proposed to integrate edge attributes into GCNs. However, they do not work when edge attributes are (partially) unavailable. To address this issue, we propose **EE-GCN**, a generic edge-empowered framework that achieves node-edge enhancement, regardless of the availability of edge attributes. In **EE-GCN** framework, three key components (*i.e.*, initialization, feature space alignment, and node-edge mutually enhanced updating) are designed and introduced to address the corresponding challenges in the learning process. We evaluate the proposed framework on different types of real-world datasets (with all/partial/no edge attributes available). The extensive experimental results demonstrate the effectiveness of our proposed framework in various of tasks including node classification, link prediction, and graph classification.

REFERENCES

[1] James Atwood and Don Towsley. 2016. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems*. 1993–2001.

[2] Joost Bastings, Ivan Titov, Wilker Aziz, Diego Marcheggiani, and Khalil Sima'an. 2017. Graph convolutional encoders for syntax-aware neural machine translation. *arXiv preprint arXiv:1704.04675* (2017).

[3] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261* (2018).

[4] Yoshua Bengio et al. 2009. Learning deep architectures for AI. *Foundations and trends® in Machine Learning* 2, 1 (2009), 1–127.

[5] Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence* 35, 8 (2013), 1798–1828.

[6] Yoshua Bengio and Olivier Delalleau. 2009. Justifying and generalizing contrastive divergence. *Neural computation* 21, 6 (2009), 1601–1621.

[7] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. 2007. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*. 153–160.

[8] Antoine Bordes, Jason Weston, Ronan Collobert, and Yoshua Bengio. 2011. Learning structured embeddings of knowledge bases. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*.

[9] Matthew Brand. 2003. Charting a manifold. In *Proceedings of the 16th annual conference on neural information processing systems (NIPS'03)*. 985–992.

[10] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*. 3844–3852.

[11] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*. 2224–2232.

[12] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 1263–1272.

[13] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*. 1024–1034.

[14] Frank Harary and Robert Z Norman. 1960. Some properties of line digraphs. *Rendiconti del Circolo Matematico di Palermo* 9, 2 (1960), 161–168.

[15] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. 2006. A fast learning algorithm for deep belief nets. *Neural computation* 18, 7 (2006), 1527–1554.

[16] Geoffrey E Hinton and Richard S Zemel. 1994. Autoencoders, minimum description length and Helmholtz free energy. In *Advances in neural information processing systems*. 3–10.

[17] Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. 2016. Molecular graph convolutions: moving beyond fingerprints. *Journal of computer-aided molecular design* 30, 8 (2016), 595–608.

[18] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[19] Nils Kriege and Petra Mutzel. 2012. Subgraph matching kernels for attributed graphs. *arXiv preprint arXiv:1206.6483* (2012).

[20] Hugo Larochelle and Yoshua Bengio. 2008. Classification using discriminative restricted Boltzmann machines. In *Proceedings of the 25th international conference on Machine learning*. ACM, 536–543.

[21] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. 2009. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th annual international conference on machine learning*. ACM, 609–616.

[22] Ruoyu Li, Sheng Wang, Feiyun Zhu, and Junzhou Huang. 2018. Adaptive graph convolutional neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[23] Yikang Li, Wanli Ouyang, Bolei Zhou, Jianping Shi, Chao Zhang, and Xiaogang Wang. 2018. Factorizable net: an efficient subgraph-based framework for scene graph generation. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 335–351.

[24] Ziyao Li, Liang Zhang, and Guojie Song. 2019. GCN-LASE: Towards Adequately Incorporating Link Attributes in Graph Convolutional Networks. *arXiv preprint arXiv:1902.09817* (2019).

[25] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. 2015. Learning entity and relation embeddings for knowledge graph completion. In *Twenty-ninth AAAI conference on artificial intelligence*.

[26] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of Machine Learning Research* 9, Nov (2008), 2579–2605.

[27] Diego Marcheggiani, Joost Bastings, and Ivan Titov. 2018. Exploiting semantics in neural machine translation with graph convolutional networks. *arXiv preprint arXiv:1804.08313* (2018).

[28] Diego Marcheggiani and Ivan Titov. 2017. Encoding sentences with graph convolutional networks for semantic role labeling. *arXiv preprint arXiv:1703.04826* (2017).

[29] Alessio Micheli. 2009. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks* 20, 3 (2009), 498–511.

[30] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. 2019. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 4602–4609.

[31] Marion Neumann, Plinio Moreno, Laura Antanas, Roman Garnett, and Kristian Kersting. 2013. Graph kernels for object category prediction in task-dependent robot grasping. In *Online Proceedings of the Eleventh Workshop on Mining and Learning with Graphs*. 0–6.

[32] Christopher Poultney, Sumit Chopra, Yann L Cun, et al. 2007. Efficient learning of sparse representations with an energy-based model. In *Advances in neural information processing systems*. 1137–1144.

[33] Sam T Roweis and Lawrence K Saul. 2000. Nonlinear dimensionality reduction by locally linear embedding. *science* 290, 5500 (2000), 2323–2326.

[34] Ruslan Salakhutdinov and Geoffrey Hinton. 2009. Deep boltzmann machines. In *Artificial Intelligence and Statistics*. 448–455.

[35] Kristof T Schütt, Farhad Arbabzadah, Stefan Chmiela, Klaus R Müller, and Alexandre Tkatchenko. 2017. Quantum-chemical insights from deep tensor neural networks. *Nature communications* 8 (2017), 13890.

[36] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. 2013. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE signal processing magazine* 30, 3 (2013), 83–98.

[37] Jeffrey J Sutherland, Lee A O'brien, and Donald F Weaver. 2003. Spline-fitting with a genetic algorithm: A method for developing classification structure- activity relationships. *Journal of chemical information and computer sciences* 43, 6 (2003), 1906–1915.

[38] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[39] Shen Wang, Zhengzhang Chen, Ding Li, Zhichun Li, Lu-An Tang, Jingchao Ni, Junghwan Rhee, Haifeng Chen, and Philip S. Yu. 2019. Attentional Heterogeneous Graph Neural Network: Application to Program Reidentification. In *Proceedings of the 2019 SIAM International Conference on Data Mining, SDM 2019, Calgary, Alberta, Canada, May 2-4, 2019*. 693–701.

[40] Shen Wang, Zhengzhang Chen, Xiao Yu, Ding Li, Jingchao Ni, Lu-An Tang, Jiaping Gui, Zhichun Li, Haifeng Chen, and Philip S. Yu. 2019. Heterogeneous Graph Matching Networks for Unknown Malware Detection. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*. 3762–3770.

[41] Yue Wang, Dawei Yin, Luo Jie, Pengyuan Wang, Makoto Yamada, Yi Chang, and Qiaozhu Mei. 2016. Beyond ranking: Optimizing whole-page presentation. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*. ACM, 103–112.

[42] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge graph embedding by translating on hyperplanes. In *Twenty-Eighth AAAI conference on artificial intelligence*.

[43] Kilian Q Weinberger and Lawrence K Saul. 2006. Unsupervised learning of image manifolds by semidefinite programming. *International journal of computer vision* 70, 1 (2006), 77–90.

[44] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. 2019. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596* (2019).

[45] Danfei Xu, Yuke Zhu, Christopher B Choy, and Li Fei-Fei. 2017. Scene graph generation by iterative message passing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5410–5419.

[46] Jianwei Yang, Jiasen Lu, Stefan Lee, Dhruv Batra, and Devi Parikh. 2018. Graph r-cnn for scene graph generation. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 670–685.

[47] LE Yann. 1987. *Modèles connexionnistes de l'apprentissage*. Ph.D. Dissertation. These de Doctorat, Universite Paris 6.

[48] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*. 5165–5175.

[49] Chenyi Zhuang and Qiang Ma. 2018. Dual graph convolutional networks for graph-based semi-supervised classification. In *Proceedings of the 2018 World Wide Web Conference*. International World Wide Web Conferences Steering Committee, 499–508.

[50] Will Y Zou, Andrew Y Ng, and Kai Yu. 2011. Unsupervised learning of visual invariance with temporal coherence. In *NIPS 2011 workshop on deep learning and unsupervised feature learning*, Vol. 3.