Property Graph Representation Learning for Node Classification

Shu Li[1,2], Nayyar A. Zaidi[3], Meijie Du[1,2], Zhou Zhou[1,2], Hongfei Zhang[1,2] and Gang Li[4*]

[1]Institute of Information Engineering, Chinese Academy of Sciences, Beijing, 100093, China.
[2]National Engineering Laboratory of Information Security Technologies, Beijing, 100093, China.
[3]School of Information Technology, Deakin University, Geelong VIC, 3216, Australia.
[4]Centre for Cyber Security Research and Innovation, Deakin University, Geelong VIC, 3216, Australia.

*Corresponding author(s). E-mail(s): gang.li@deakin.edu.au;
Contributing authors: lishu@iie.ac.cn; nayyar.zaidi@deakin.edu.au; dumeijie@iie.ac.cn; zhouzhou@iie.ac.cn; zhanghongfei@iie.ac.cn;

Abstract

Graph representation learning (graph embedding) has led to breakthrough results in various machine learning graph-based applications such as node classification, link prediction and recommendation, etc. Many real-world graphs can be characterized as the property graphs, because besides the structure information, there exists rich property information related to each node in the graphs. Many existing graph representation learning methods – e.g., random walk-based methods like `DeepWalk` and `Node2vec`, focus only on the structure of graph for learning the node embedding. Although graph representation learning based on neural networks (e.g., typical `GNN` methods such as `GraphSAGE`) use the property of nodes as the initial features of nodes and then aggregate feature information of the neighbours, their limitation is that the neighbourhood of a node is considered to be uniform – i.e., there is no way to differentiate among neighbours of a node when learning a node embedding. Additionally, their definition of neighbourhood is local, i.e., only nodes connected to the current node are considered as neighbours. Hence those methods fail to capture implicit/latent relationships among nodes, which are implicit in the given structure. In this study, our aim is to improve the performance of graph representation learning methods on property graphs. We present a new framework called `Enhanced Property Graph Embedding` (`EPGE`) – a graph representation learning framework to address above-mentioned limitations. Our proposed framework relies on the notion of latent neighbourhood, as well as systematic sampling of neighbouring nodes to obtain better representation of the nodes. The experimental results on five publicly available graph datasets demonstrate that `EPGE` outperforms state-of-the-art baselines for the task of node classification. We further evaluate the superiority of our proposed formulation by defining a novel quantitative metric to measure the usefulness of the sampled neighbourhood in the graph.

Keywords: Property Graph, Graph Representation Learning, GraphSAGE, Biased Sampling, Latent Connection

1 Introduction

Graphs are powerful data structures that allow us to easily express connectivity (in form of edges) among entities (known as nodes). Real-world graphs are ubiquitous, e.g., they can be in the form of social networks [1, 2], biological networks [3, 4], knowledge graphs [5], publication citations networks [2, 6], etc. There have been many machine learning applications on graphs, e.g., determining a community a person belongs to on an online social network, classifying the functional role of a molecule in a biological interaction graph, or predicting purchase patterns in buyers-products-sellers graphs in online e-commerce platforms, etc. Graph representation learning in machine learning – also known as graph embedding – has been proposed to encode the structural information of the graph by constructing an embedding vector for each node in graph as the node's representation. In other words, they map any node in the graph to a low-dimensional Euclidean space. Of course, the goal of graph representation learning is to optimize this mapping so that geometric relationships in this learned space reflect the structure of the original graph. Such graph embedding [7, 8] has achieved great successes in machine learning tasks, such as node classification, link prediction [1, 6, 9, 10].

Graph representation learning involves incorporating structural information of the graph. However, in practice, graphs not only contain the structure information but also contain *properties* (also called attributes) of nodes. For example, a publication citation network consists of papers as nodes and citation relationships as edges. Here, the nodes have properties related to the content of the papers. Similarly, the social network (such as `Twitter`) can be represented by a graph, in which every user is represented by a node with properties such as user profile, user behaviours, etc. Of course, the follower/followed relationships among nodes are the structural information depicted with edges. We call graphs as property graphs, in which other than the structural information, there are certain properties associated with each node. In the real-world network, there are two type of property graphs. One is homophilous graphs in which most connected nodes are from the same class or with similar properties, such as citation networks where papers mostly cite research from the same research area, and social networks where users tend to connect to users with similar interests. By contrast, graph with heterophily describes the preference of nodes to connect to nodes not similar to them. Heterophilous graph often occurs in financial transaction networks where fraudsters often perform transactions with non-fraudulent users. In dating networks, most connections are between people of opposite genders and this is also an example of heterophilous graph [11]. Note, in our study, we focus on homophilous property graphs, i.e., edges in graphs tend to connect to similar nodes.

Among the existing work of graph representation learning, `Graph Neural Networks` (`GNN`) are undoubtedly the most effective to model the property graphs with homophily. Conceptually, the fundamental idea of `GNN` models is to employ deep artificial neural networks to learn an embedding of each individual node in the graph by aggregating not only feature information of the node but as well as its local neighbourhood. For example, `Graph Convolutional Networks` (`GCN`) [6] – an example of `GNN`, performs convolution on graph by transforming node representations into the spectral domain using the graph Fourier transform. `GraphSAGE` [1] extends `GCN` from a spectral method to a spatial one and can efficiently generate node embeddings for previously unseen data by sampling and aggregating features from a node's local neighbourhood. Such `GNN` have shown remarkable performance in many downstream tasks especially on graph with good homophily nature [12]. Despite the power of these `GNN` methods, they have certain limitations:

- They are likely to be ineffective in aggregating neighbouring information for nodes that have no or few relations with other nodes in the graph.
- The neighbourhood of a node is defined as the set of all neighbours which are one or more hops away. We conjecture that there might be nodes which could be very similar to the node in question but are not in its neighbourhood. Existing methods are not able to aggregate such highly similar (non-neighbourhood) nodes.
- Spatial-based methods like `GraphSAGE` [1], sample all neighbours equally when aggregating their information. It does not consider the fact that different neighbours may influence the node rather differently.

To address the above limitations, in this paper, we propose a novel framework named `Enhanced Property Graph Embedding` (`EPGE`) – for graph representation learning in a property graph. `EPGE` has the following salient features:

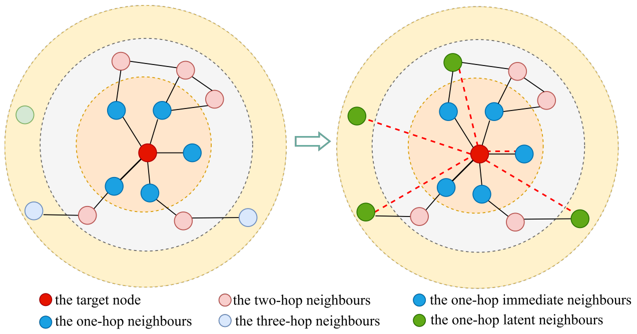


Fig. 1 Illustration of *Latent Graph*. The original graph is on the left, and we create a latent graph on the right. Dash lines represent latent connections in which the new neighbours of the red node are constructed based on the node property similarity.

- To address the first and second challenges described above, apart from the existing (original) graph, we create a latent graph based on the node property information (illustrated in Figure 1). This will help drastically for nodes with none or few neighbours in the original graph. Moreover, the latent graph has the ability to capture the important features from distant but informative nodes.
- To address the third challenge, a bias strategy is applied to sample neighbours (not only immediate neighbours but latent neighbours) for differentiating the influences of the neighbours. We have proposed a sampling strategy to help choose the most informative neighbours.

Once the neighbours are selected, we aggregate the immediate and latent neighbourhoods to compute the final node embeddings. We claim that the final embeddings obtained with `EPGE` are much more powerful than existing state-of-the-art methods. Additionally, to back up this claim, a quantitative evaluation metric is defined to measure the usefulness of sampled neighbourhood information. By conducting experiments on five public graph datasets, we demonstrate the superior performance of `EPGE` over existing state-of-the-art baselines. Moreover, we separately validate the efficacy and performance of the salient features of `EPGE` by performing detailed ablation studies. We also discuss how various parameters (e.g., the parameter of latent connection construction, the size of sampling neighbours set, and the number of the edges) impact the model performance.

Our contributions are summarized as follows:

- We propose a method for property graph representation learning, which not only exploits the existing graph, but also builds a latent graph. In addition, it has an effective neighbourhood sampling technique.
- We define a quantitative metric value to measure the usefulness of the sampled neighbourhood, which helps evaluate the superiority of our proposed method.

- We report an extensive experimental analysis to evaluate the merits of our proposed algorithms.

The rest of the paper is organized as follows. Section 2 reviews existing work on graph representation learning. Section 3 provides the preliminaries followed by Section 4 that presents the proposed method. Section 5 defines a quantitative index to evaluate and explain the effectiveness of the proposed `EPGE` model. We discuss experimental results in Section 6, and conclude with our contributions and future work in Section 7.

2 Related Work

A growing body of literature has been devoted to graph representation learning. In the following, let us discuss a few prominent lines of direction.

2.1 Random Walk-based Methods

Random walk-based methods are one of the early approaches to graph representation learning that approximate various characteristics such as node centrality [13] and similarity [14] of the graph.

Two prominent examples of graph embedding techniques based on random walk are `DeepWalk` [15] and `Node2vec` [16]. `DeepWalk` [15] proposed to use a skip-gram model to learn node embeddings by constructing the relationships among nodes based on paths obtained from random walks. It was found statistically that the frequency that nodes appear in the short random walks will follow a power-law distribution as the word frequency in natural language. Therefore, language modeling can be applied to graph representation learning. `DeepWalk` presented a generalization of language modeling to explore the graph through a stream of short random walks. These walks can be thought of as short sentences and nodes in walks are analogous to words in sentences. `Node2vec` [16] utilized biased random walks by reaching a trade-off between breadth-first and depth-first graph search. Specifically, `Node2vec` introduced two hyper-parameters ρ and q to control the breadth-first search and depth-first search in the random walk, respectively. Grid search is used to seek the optimal hyper-parameters for network representation learning. Therefore, `Node2vec` can not only obtain local topology information of the node but also explore deeper structural information, thereby improving the effectiveness of network representation learning. `Walklets` [17] modified the random walk strategy in `DeepWalk`. By skipping over steps in each random walk, `Walklets` generated a corpus of node pairs that are reachable via paths of a fixed length. This corpus can then be used to learn a series of latent representations, each of which captures successively higher-order relationships from the adjacency matrix. By this random walk strategy, `Walklets` can capture the relationship among nodes with a larger spatial scale. `HARP` [18] utilized a graph coarsening procedure to collapse related nodes in the graph together into super nodes. This coarsened graph was used to learn a set of initial representations, and the learned embedding of each super node was used as an initial value for the random walk embeddings of the super node's constituent nodes. The process can repeat in a hierarchical manner at varying levels of coarseness. In [7], it was shown that random walks-based methods are inefficient for processing large graphs, because the node embeddings are independent and there is no sharing of parameters. In addition, only the structure information of the graph is learned and the properties of nodes are not taken into consideration in such models.

2.2 Graph Neural Networks

Growing research in deep learning over the past few years has led to a deluge of deep neural networks based methods applied to graphs [1, 6, 19], leading to a formulation known as the `Graph Neural Networks` (`GNN`). Unlike random walk-based methods, `GNN` encode nodes into vectors by aggregating feature information from node's local neighbourhood via neural networks.

Several researchers have attempted to define convolutions operators on graph to learn graph presentation. Graph convolutions can often be categorized as spectral approaches and spatial approaches. Spectral approaches perform convolution by transforming node representations into the spectral domain using the graph Fourier transform or its extensions. [20] first introduced convolution for graph data from the spectral domain using the graph Laplacian matrix, and used a learnable diagonal matrix as the filter. However, this operation in [20] is computationally inefficient and the filter is non-spatially localized. To solve this efficiency problem, [21] proposed the ChebNet and improved the spectral-based approach by using a polynomial filter. [6] further simplified the filtering by using only the first-order neighbours. Spatial approaches perform convolutions directly on the graph based on the graph topology. The major challenge of spatial approaches is defining the convolution operation with differently sized neighborhoods. [22] proposed a spatial method that used different weight matrices for nodes with different degrees, but it may not be scalable to large scale graphs with more node degrees. [23] used transition matrices to define the neighborhood for nodes. [24] defined a "receptive field" for each node by selecting a fixed number of nodes from its k-step neighborhoods, and adopted a standard 1-D CNN with proper normalization to learn grap embedding.

Many techniques have been introduced to further improve `GNN`, especially in spatial approaches, and some of these methods are general. Inspired by the attention mechanism, [10] incorporated the attention mechanism into `GNN` so that the node neighborhoods are aggregated with different weights. Some methods added "skip connections" to make `GNN` models deeper. In [2], researchers explored an architecture that learned to selectively exploit information from neighborhood of differing locality. They proposed the `Jumping Knowledge Networks` that selectively combines different aggregations at the last layer, i.e. the node representations of each layer directly "jumps" to the last layer. This network learned the representations of different orders for different graph substructures, hence the trained model can improve the representations, in particular for graphs with sub-graphs of diverse local structures. In the area of computer vision, a convolutional layer is usually followed by a pooling layer to get more general features. Similar to these pooling layers, some research focuses on designing hierarchical pooling layers on graphs. H-GCN [25] repeatedly aggregated nodes with similar structures to form some hyper-nodes, followed by refining the coarsened graph to the original with an aim to restore the representation for each node. Instead of merely aggregating one or two-hop neighbourhood information, the proposed coarsening procedure enlarged the receptive field for each node, hence more global information can be captured. GNN models aggregate messages for each node from its neighborhood. Intuitively, if multiple GNN layers are implemented, the size of neighbours will grow exponentially with the depth. Therefore, a sampling technique is adopted to alleviate this "neighbour explosion" issue. GraphSAGE [1] was able to efficiently generate node embeddings for previously unseen data by sampling and aggregating features from a node's local neighbourhood. GraphSAGE [1] does not utilize the full set of neighbours but a fixed-size set of neighbours by uniformly sampling. Our method is based on GraphSAGE, and proposes an effective neighbour sampling technique.

In addition to the above techniques in GNN, recent researchers have attempted to build GNN by designing and optimizing the network architecture. DAGNN [26] proposed a deep adaptive graph neural network to learn node representations from larger receptive fields. N-GCN [27] trained multiple instances of GCN over node pairs discovered at different distances in random walks, and learned a combination of the instance outputs. In fact, our proposed method can also obtain larger receptive fields by introducing the latent neighbours that are similar but are far away from each other.

More recent attention has been focused on GNN models on graphs with heterophily, where most connected nodes are from different classes. [28] proposed a framework called CPGNN that incorporated an interpretable compatibility matrix for modelling the heterophily or homophily level in the graph, enabling it to go beyond the assumption of strong homophily. [12] designed a propagation mechanism, which can automatically change the propagation and aggregation process according to homophily or heterophily between node pairs. By introducing two measurements of homophily degree, this model can adaptively learn the propagation process. [29] proposed a GNN model based on a bi-kernel feature transformation and a selection gate. Two kernels capture homophily and heterophily information respectively, and the gate is introduced to select which kernel is used for the given node pairs. [11] declared that the most significant drawback of the standard datasets used for evaluating heterophily-specific models is the presence of a large number of duplicate nodes in the datasets, leading to train-test data leakage and making results obtained by using these datasets unreliable. [11] showed that standard GNN achieved strong results on heterophilous graphs, almost always outperforming specialized models. Compared with these studies that automatically learn the homophily or heterophily between node pairs to improve the performance of standard GNN models under heterophily, our proposed framework EPGE focus on homophilous graphs that aims to capture the similarities not only from the local neighbours but also the explicit, i.e., latent node to enhance node representation to the maximum extent.

3 Preliminaries

Let us discuss some preliminaries in this section.

3.1 Graph Definition

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ represents an undirected graph with N nodes and their connection edges. We have a set of nodes – $v_i \in \mathcal{V}$, and edges $(v_i, v_j) \in \mathcal{E}$. The features of nodes are denoted as $\mathbf{Z} = \{\mathbf{z}_1, ..., \mathbf{z}_N\} \in R^{N \times F}$ where F denotes the size of the feature vector. For any node $v_i \in \mathcal{V}$, $\mathcal{N}(v)$ is the set of nodes that are in the neighbourhood of node v based on $\mathcal{E}$.

3.2 Graph Representation Learning

By definition, graph representation learning (graph embedding) is an approach that learns a mapping from high-dimensional sparse graphs into low-dimensional, dense and continuous vector spaces, while maximally preserving the graph structure properties. Graph embedding has been used in the literature in two ways [4]:

- `Graph Embedding` encodes each node of a graph with its own vector representations with a smaller dimension, with the following definition:

  Definition 1 (Graph Embedding) Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, having nodes $\mathcal{V}$ and their connecting edges $\mathcal{E}$. Graph Embedding is a mapping $f : v_i \in \mathcal{V} \to \mathbf{y}_i \in \mathbb{R}^d$, such that $d \ll |\mathcal{V}|$ and the function f preserves some proximity measure, like node similarity in the original graph $\mathcal{G}$.

- `Whole-graph Embedding` is to represent the whole graph in the form of latent vectors, and defined as:

  Definition 2 (Whole-graph Embedding) Given a set of graphs $\mathcal{G} = \{\mathcal{G}_1, ..., \mathcal{G}_m\}$, whole-graph embedding is a mapping $f : \mathcal{G}_i \to \mathbf{y}_i \in \mathbb{R}^d$, such that the function f preserves some proximity measure defined on graph $\mathcal{G}$.

In this study, we use the former definition. As thus, an embedding maps each node to a low-dimensional feature vector, and such generated nonlinear and highly informative graph embeddings can be conveniently used to address the node classification task.

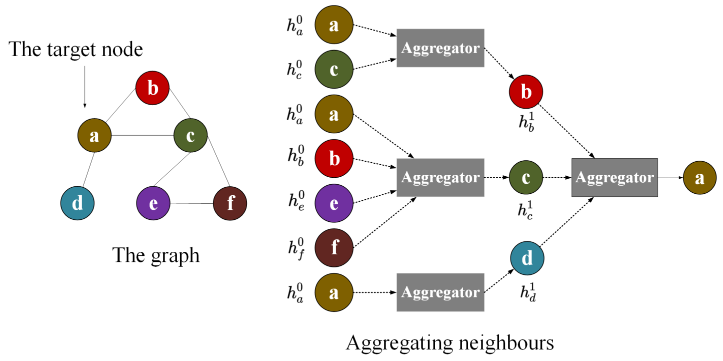


Fig. 2 Aggregators in the `GraphSAGE`.

3.3 GraphSAGE

As we discussed earlier, `GraphSAGE` [1] is an improvement over the original `GCN` model. `GCN` is trained independently for a fixed graph and requires full graph Laplacian, so it is inherently transductive learning. `GraphSAGE` replaces the full graph Laplacian with learnable aggregation functions which are key to performing message passing, and it is a general inductive learning framework that can efficiently generate the embedding of unknown nodes by using the feature information of nodes. Such ability for inductive learning is important for processing large-scale graphs, leading to strong generalization performance to unseen nodes.

The core idea of `GraphSAGE` is to generate embeddings of the target node by learning an aggregator function that samples and aggregates features from a node's local neighbourhood, as shown in Figure 2. The sampling strategy used in `GraphSAGE` is to uniformly sample a fixed-size set of neighbours and sample with replacement in case where the sample size is larger than the node's degree. Five candidate aggregator functions are used namely

1. `MEAN aggregator`,
2. `GCN aggregator`,
3. `LSTM aggregator`,
4. `MeanPooling aggregator`, and
5. `MaxPooling aggregator`.

Let us discuss these five aggregators at k-th depth in the following.

`MEAN aggregator` is defined as:

$$\begin{aligned}\mathbf{h}^k_{\mathcal{N}(v)} &\leftarrow \text{MEAN}\Big(\{\mathbf{h}^{k-1}_u, \forall u \in \mathcal{N}(v)\}\Big);\\ \mathbf{h}^k_v &\leftarrow \sigma\Big(\mathbf{W}\cdot \text{CONCAT}(\mathbf{h}^{k-1}_v, \mathbf{h}^k_{\mathcal{N}(v)})\Big).\end{aligned} \tag{1}$$

$\mathcal{N}(v)$ is the immediate neighbourhood set of node v. $\mathbf{h}$ denotes a node's representation at this step. MEAN is the mean operator, where the element-wise mean of the vectors $\mathbf{h}^k_{\mathcal{N}(v)}$ is taken. The immediate neighbourhood is aggregated into a single vector $\mathbf{h}^k_{\mathcal{N}(v)}$, and `GraphSAGE` then concatenates this aggregated neighborhood vector with the node's current representation $\mathbf{h}^{k-1}_v$. $\mathbf{W}$ denotes the weight matrix, and σ is a non-linear activation function.

On the other hand, `GCN aggregator` is defined as:

$$\mathbf{h}^k_v \leftarrow \sigma\Big(\mathbf{W} \cdot \text{MEAN}(\{\mathbf{h}^{k-1}_v\} \cup \{\mathbf{h}^{k-1}_u, \forall u \in \mathcal{N}(v)\}\Big), \tag{2}$$

where $\cup$ is the union operation.

`LSTM aggregator` replaces the MEAN operation in Equation 1 as:

$$\mathbf{h}^k_{\mathcal{N}(v)} \leftarrow \text{LSTM}\Big(\{\mathbf{h}^{k-1}_u, \forall u \in \pi(\mathcal{N}(v))\}\Big), \tag{3}$$

where LSTM (Long Short-term Memory) is a special `Recurrent Neural Network` to process inputs in a sequential manner. $\pi(\cdot)$ is a random permutation operation.

`MeanPooling aggregator` replaces the MEAN operation in Equation 1 as:

$$\mathbf{h}^k_{\mathcal{N}(v)} \leftarrow \text{MEAN}\Big(\{\sigma(\mathbf{W}_{\text{pool}}\mathbf{h}^{k-1}_{u_i} + \mathbf{b}), \forall u_i \in \mathcal{N}(v)\}\Big). \tag{4}$$

Here, $\mathbf{W}_{\text{pool}}$ is the weight matrix of a fully-connected layer in this pooling aggregator.

Finally, `MaxPooling aggregator` replaces the MEAN operation in Equation 1 as:

$$\mathbf{h}^k_{\mathcal{N}(v)} \leftarrow \text{MAX}\Big(\{\sigma(\mathbf{W}_{\text{pool}}\mathbf{h}^{k-1}_{u_i} + \mathbf{b}), \forall u_i \in \mathcal{N}(v)\}\Big). \tag{5}$$

`GraphSAGE` aims at learning an aggregator instead of learning a representation for each node. This idea can improve the flexibility and generalization ability of the model. In addition, thanks to its flexibility, it can train the model in batches to improve the convergence speed. It is important to note that, `GraphSAGE` consistently outperforms state-of-the-art baselines in `GNN` [1]. Although `GraphSAGE` and other existing neighbourhood aggregation methods have achieved good performance on graph representation learning, they are not able to aggregate nodes that are similar but are far away from each other. Moreover, these methods overlook the fact that different nodes in the neighbourhood may have different influences on the node. We will address these issues with our proposed `EPGE` method, and further improve the performance of `GraphSAGE` in this study.

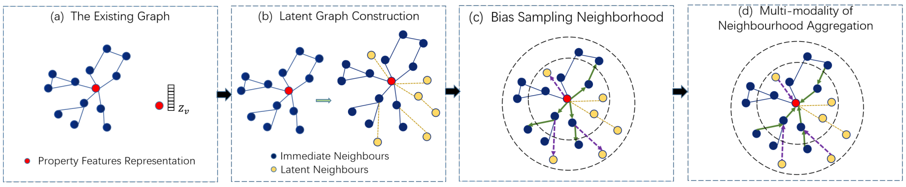

(a) The Existing Graph
z_v
● Property Features Representation
(b) Latent Graph Construction
● Immediate Neighbours
● Latent Neighbours
(c) Bias Sampling Neighborhood
(d) Multi-modality of Neighbourhood Aggregation

Fig. 3 Pictorial illustration of the of `EPGE` framework.

3.4 Existing and Latent Graphs

We define *Existing Graph* as:

Definition 3 (Existing Graph) An existing graph is a graph that can be created based on original connections between entities, and is parametrized as: $\mathcal{G}^e = (\mathcal{V}, \mathcal{E}_{\mathcal{G}^e}, \mathcal{P})$, where $\mathcal{V}$ are the nodes, $\mathcal{E}_{\mathcal{G}^e}$ correspond to their relationships, and $\mathcal{P}$ denotes the properties of nodes.

The existing graph defined above is fundamental to describe the relationship of the nodes, in which the node representation can be efficiently improved by aggregating features from its neighbours. However, it is unable to capture the long-distance dependencies between nodes with similar properties when they are far away in the existing graph. In order to address this issue, we define *Latent Graph* as:

Definition 4 (Latent Graph) A latent graph is parametrized as: $\mathcal{G}^l = (\mathcal{V}, \mathcal{E}_{\mathcal{G}^l}, \mathcal{P}, \lambda_{\mathcal{G}^l})$ with nodes $\mathcal{V}$ and links $\mathcal{E}_{\mathcal{G}^l}$, where the link between two nodes $u, v \in \mathcal{V}$ exists if the similarity between the nodes exceeds a certain pre-defined threshold $\lambda_{\mathcal{G}^l}$. $\mathcal{P}$ represents the property of nodes as the same meaning in the $\mathcal{G}^e$.

It can be seen that edges in the latent graph depend on the similarity between nodes and are dominated by a pre-defined threshold. Of course, one can control the number of edges by changing this threshold.

4 Methodology

Let us discuss our proposed `EPGE` framework in this section. We will start by discussing the framework, followed by our discussion of latent graph construction. Later, we will discuss various features of our proposed formulation.

4.1 `EPGE` Framework

Our proposed framework is illustrated in Figure 3 which depicts various steps in our formulation. First, the existing graph $\mathcal{G}^e$ is obtained along with the property vector of nodes. Second, two nodes are similar if the similarity of their property vectors exceeds a pre-defined threshold $\lambda_{\mathcal{G}^l}$, leading to a latent edge between the two nodes. These latent edges and the corresponding nodes form the latent graph $\mathcal{G}^l$. Third, a biased neighbourhood sampling strategy is implemented. Neighbours that are more similar to the node have higher priority to be aggregated until a fixed-size set of neighbours is obtained. Note, in this step, the neighbourhood contains the immediate neighbours in the existing graph and the latent neighbours in the latent graph. Finally, a node embedding $\mathbf{x}_v$ is obtained by aggregating the property vectors of the neighbours and itself.

4.1.1 Latent Graph Construction

Let us discuss the creation of the latent graph. For the existing property graph $\mathcal{G}^e$, we have, nodes $v_i \in \mathcal{V}$ and edges $(v_i, v_j) \in \mathcal{E}_{\mathcal{G}^e}$. The property vector representation of node $v \in \mathcal{V}$ is denoted as $\mathbf{z}_v$ [1]. For $v, u \in \mathcal{V}$, their similarity and the latent edge are defined as:

$$\begin{aligned} S(v,u) &= \mathrm{PearsonSimilarity}(\mathbf{z}_v, \mathbf{z}_u); \\ \mathcal{E}_{\mathcal{G}^l}(u,v) &= \begin{cases} 1, & \text{if } S(u,v) \geq \lambda_{\mathcal{G}^l} \\ 0, & \text{otherwise.} \end{cases} \end{aligned} \tag{6}$$

Here $S(u,v)$ denotes the similarity between two nodes – if it exceeds the threshold $\lambda_{\mathcal{G}^l}$, these two nodes are linked by a latent edge, which finally creates a latent graph $\mathcal{G}^l$. One can use any form of similarity measure such as: `Pearson, Spearman, dot product`, etc. Here, we adopt `Pearson` as the default measure as it works the best in practice. Importantly, we analyse the effect of the parameter $\lambda_{\mathcal{G}^l}$ setting for the model performance in Section 6.7.1.

4.1.2 Bias Sampling Neighborhood

Many real-world graphs have high-degree nodes, i.e., nodes with a large number of neighbours. Considering all neighbours for aggregation is usually inefficient and unnecessary [30]. Given that a node's neighbours in a graph have no natural ordering, though they do in sentences, images, etc., `GraphSAGE` [1] uniformly samples a fixed-size set of neighbours. It has been demonstrated that aggregating neighbour information is effective in graph representation learning [1]. However, not all neighbours of a node can provide positive information. Therefore, our proposed framework `EPGE` improves `GraphSAGE` by deriving a set of sampled neighbours based on their similarity. The intuition is that similar neighbours (similar in any type of property) can consolidate and enhance the node embedding results. In other words, in `EPGE` model, neighbours that are more similar to the node being processed have higher priority to be aggregated. We will discuss how to evaluate the benefit of this strategy in Section 5 and provide the experimental analysis in Section 6.8.

[1]Note, the properties of nodes depend on specific scenarios. For example, the properties of nodes in a publication citation network include titles, authors, year of publication, the content of the papers, etc. For another instance, the properties of user nodes in a social network constitute user's profile, user's behaviours, user's posts and so on.

4.1.3 Multi-modality Neighbourhood Aggregation

The neighbourhood $\mathcal{N}(v) = \left\{\mathcal{N}_e(v), \mathcal{N}_l(v)\right\}$ of node v includes its neighbourhoods in both the existing graph and the latent graph. The existing-neighbourhood $\mathcal{N}_e(v)$ consists of the set of v's adjacent nodes in the existing graph $\mathcal{G}^e$, and the latent-neighbourhood $\mathcal{N}_l(v)$ are those whose similarity to node v is higher than a parameter $\lambda_{\mathcal{G}^l}$. We regard these two types of neighbourhood as multi-modality neighbourhood and discuss the fusion of multi-modality neighbourhood in this section. During the process of aggregation, we combine the existing neighbourhood and the latent neighbourhood to generate the node embeddings. The motivation is that different types of neighbours will make different contributions to the final node representations. For the existing-neighbourhood, it denotes the effect of user's original nature. In comparison to this easily presented relationship, the latent-neighbourhood indicates the long-range dependencies with the node, which is invisible and cannot be captured directly. Here, we have two approaches to aggregating the existing neighbourhood and the latent neighbourhood. One is to treat these two modalities of neighbourhood equally and sample the neighbours completely according to the similarity with the node in problem. Another approach is that the existing neighbourhood is given priority to be sampled, while the latent neighbourhood is regarded as a supplement until a fixed-size set of neighbours is obtained. In the experiment part, we adopt the later approach.

4.2 The Algorithm of EPGE

Algorithm 1 describes the overall procedure of our proposed EPGE framework. From step 3 to 9, the similarity of nodes is calculated, and the latent neighbours are constructed based on similarity. In step 11, when the size of the existing neighbourhood is less than β (the predetermined size of neighbours to be sampled), we select $\beta - |\mathcal{N}_e(v)|$ neighbours from the latent neighbours as supplement. The latent neighbours are sorted to obtain the neighbours with the highest similarity. Next, for each node $v \in \mathcal{V}$, it aggregates the representations of its sampled neighbourhood, $\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}^s(v)\}$, represented in step 15. Here, AGGREGATOR is one of the five candidate aggregator functions introduced in Section 3.3. Then we use CONCAT operation to concatenate the node's sampled neighbourhood $\mathbf{h}^k_{\mathcal{N}^s(v)}$ and its current representation $\mathbf{h}_v^{k-1}$ (Step 16). This concatenated vector is fed through a fully connected layer with a non-linear activation function σ. In this process, we can adopt the five aggregators described in Section 3.3. Finally, we get the final representations output at depth K, denoted as $\mathbf{x}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$ (Step 21). According to the learned nodes representation, the classification task of graph nodes can be conducted.

4.3 Model Training

In this study, we take node classification in a supervised setting as the specific downstream task. The cross-entropy is applied as the loss function of

Algorithm 1 The EPGE Algorithm

```
Input: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{P})$;
        Property features $\{\mathbf{z_v}, \forall v \in \mathcal{V}\}$;
        Depth K;
        Weight matrices $\mathbf{W}^k, \forall k \in \{1..., K\}$;
        Non-linearity $\sigma$;
        The immediate neighborhood $\mathcal{N}_e(v)$;
        The latent neighborhood $\mathcal{N}_l(v)$;
        All neighborhood $\mathcal{N}(v)$;
        The sampled neighbourhood $\mathcal{N}^s(v)$, the size $|\mathcal{N}^s(v)|$;
        The size of neighbours to be sampled $\beta$;
        The threshold parameter of latent connection construction $\lambda_{\mathcal{G}^l}$;
Output: Vector representations $\mathbf{x}_v$ for all $v \in \mathcal{V}$;
1:  $\mathbf{h}_v^0 \leftarrow \mathbf{z_v}, \forall v \in \mathcal{V}$;
2:
3:  for $v \in \mathcal{V}$ do
4:      for $u \in \mathcal{V}$ do
5:          if $S(v,u) \geq \lambda_{\mathcal{G}^l}$ using Equation 6 then
6:              $\mathcal{N}_l(v) \leftarrow u$ ;
7:          end if
8:      end for
9:  end for
10:
11: $\mathcal{N}^s(v) \leftarrow \Big\{\mathcal{N}_e(v) \cup \mathrm{Sort}(\mathcal{N}_l(v), \beta - |\mathcal{N}_e(v)|)\Big\}$;
12:
13: for $\{k = 1...K\}$ do
14:     for $v \in \mathcal{V}$ do
15:         $\mathbf{h}^k_{\mathcal{N}^s(v)} \leftarrow \mathrm{AGGREGATOR}_k\Big(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}^s(v)\}\Big)$ ;
16:         $\mathbf{h}_v^k \leftarrow \sigma\Big(\mathbf{W}^k \cdot \mathrm{CONCAT}\ (\mathbf{h}_v^{k-1}, \mathbf{h}^k_{\mathcal{N}^s(v)})\Big)$;
17:     end for
18:     $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \mathbf{h}_v^k{}_2, \forall v \in \mathcal{V}$ ;
19: end for
20:
21: $\mathbf{x}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$;
```

the model. With the labelled nodes, we train EPGE by minimizing the cross entropy via back-propagation and gradient descent. Thus, the loss function is calculated as:

$$\mathcal{L} = \sum_{v \in \mathcal{V}} \Big(y_v \log p_v + (1 - y_v) \log(1 - p_v)\Big), \qquad (7)$$

$$\text{where} \quad p_v = \sigma(\mathbf{w}^T \mathbf{x}_v + b).$$

This cross-entropy loss function compares the prediction results of the model with the real labels of the data in the classification task. Here, $\mathbf{x}_v$ is the final node representation of the node v obtained from Algorithm 1, p_v is the predicted probability of node v, and y_v is the ground truth.

5 Label Consistency Metric

GNN models obtain the node representation by collecting information from the neighbourhood. However, in practice, not all neighbours of a node contain relevant information, which means that some neighbours may convey positive information to the node and some neighbours may provide negative disturbance. For the node classification task, it is reasonable to consider that neighbours with the same class label of the target node can contribute positive information. In our proposed EPGE framework, based on GraphSAGE, we improve the performance by introducing the latent neighbourhood and adopting the biased sampling strategy. The purpose of adopting these two strategies is to choose neighbourhood which is helpful to node representation. In order to interpret why our proposed method can better select neighbourhood and thus achieve better performance, we introduce a metric named `label consistency`, which quantitatively measures the usefulness of the sampled neighbourhood information.

Consider the node classification task where each node $v \in \mathcal{V}$ has a label y_v, we say $v_i \simeq v_j$ if $y_{v_i} = y_{v_j}$. Then we define the label consistency metric as:

$$\tau = \sum_{e_{v_i,v_j} \in \mathcal{N}^s_{(v)}} \Big(\mathbb{I}(v_i \simeq v_j) / |\mathcal{N}^s(v)| \Big), \tag{8}$$

where $\mathbb{I}(v_i \simeq v_j)$ is an indicator function, representing the count of $v_i \simeq v_j$. $\mathcal{N}^s(v)$ is the sampled neighbourhood. A larger τ implies that neighbours with the same labels tend to be sampled, in which case the surrounding contributes more positive information for the node representation. Therefore, the larger the τ, the better the sampled neighbours for node representation learning. In the Section 6.8, we calculate this metric of EPGE and GraphSAGE on the experimental datasets, which explains, to some extent, the superiority of the proposed framework EPGE than GraphSAGE.

6 Experiment and Analysis

6.1 Datasets

We evaluate our proposed method on five public datasets, which are widely used for GNN node classification. The statistics of these datasets are summarized in Table 1. The details of these datasets are as follows:

- `Cora` [31] is citation network dataset consisting of machine learning papers as nodes and the citation relationships as edges. Those papers generate a vocabulary of 1433 unique words after stemming and removing stop-words.

Table 1 Statistic of Datasets.

Dataset	Task	Classes	Nodes	Edges	Features
Cora	multi-class	7	2,708	5,278	1,433
CiteSeer	multi-class	6	3,312	4,536	3,703
PubMed	multi-class	3	19,717	44,338	500
PPI	multi-label	121	14,755	222,109	50
HateUser	multi-class	2	4,971	9,620	320

Each paper is represented with binary values indicating whether each word in the vocabulary is *present* (indicated by 1) or *absent* (indicated by 0) in the paper.

- `CiteSeer` [31] is another citation network dataset, in which documents and citations are treated as nodes and edges. `CiteSeer` papers generate a vocabulary of 3703 unique words. The same as `Cora` Dataset, the property vectors of nodes are presented by these words.
- `PubMed` [32] is a citation network from the PubMed database, which contains a set of articles (nodes) related to diabetes and the citation relationships (edges) among them. The node properties are TF-IDF representation for the article, and the node labels are the diabetes type addressed in the articles.
- `PPI` [1] is a biological graph of `Protein-Protein Interactions` (predicting protein functions). The node's properties include positional gene set, motif set and immunological features.
- `HateUser` contains a network of $100k$ users, among them about $5k$ were annotated to be either hateful or not. If one user has retweeted the post of another user, such a retweet connection is represented as the edge in this dataset. The properties of users could be content-related, activity-related, sentiment-related, etc.

6.2 Baselines

We compare our proposed model with ten baselines of graph representation learning. `DeepWalk` [15] and `Node2vec` [16] are the representatives of random walk based methods for graph representation learning. The recent models of graph convolutional networks include `GCN` [6], `GAT` [19], `JK-LSTM` [2], `H-GCN` [25], `N-GCN` [27], `DAGNN` [26], and `Geom-GCN` [33]. `GraphSAGE` is the state-of-the-art based on neighbour aggregation. As our proposed model `EPGE` mainly makes improvement based on `GraphSAGE`, we conduct a more detailed comparative analysis between `EPGE` and `GraphSAGE`. Let us discuss the details of baseline approaches in the following:

- `DeepWalk` [15] – A skip-gram model to learn node embeddings by capturing the relationships between nodes based on random walk paths.
- `Node2vec` [16] – A method considers both graph homophily and structural equivalence by combining breadth-first random walk and depth-first random walk.

- GCN [6] – A scalable implementation of GCN via a localized first-order approximation of spectral graph convolutions.
- GAT [19] – A graph neural network architectures that assigns different importance to different neighbours by utilising self-attention mechanism, and then combines their impacts to generate node embeddings.
- JK-LSTM [2] – This model proposed the Jumping Knowledge Networks that selectively combine different aggregations at the last layer.
- H-GCN [25] – This work designed a graph coarsening layer to aggregate nodes with similar structures to hyper-nodes for enlarging the receptive field for each node and improving the performance.
- N-GCN [27] – This method trained multiple instances of GCN over node pairs discovered at different distances in random walks, and learned a combination of the instance outputs.
- DAGNN [26] – A deep adaptive graph neural network is presented by decoupling the representation transformation and propagation operations, which can ease the over-smoothing issue.
- Geom-GCN [33] – This scheme first maps a graph to a continuous latent space via node embedding, and then use the geometric relationships defined in the latent space to build structural neighborhoods for aggregation, and then design a bi-level aggregator operating on the structural neighborhoods to update the feature representations of nodes in graph neural networks.
- GraphSAGE [1] – GraphSAGE sampled and aggregated features from a node's local neighbourhood, instead of training individual embeddings for each node. GraphSAGE reported the experimental results of four aggregators, namely GraphSAGE-GCN, GraphSAGE-MEAN, GraphSAGE-LSTM, and GraphSAGE-MaxPooling.

6.3 Experiment Setup

We implement our method using TensorFlow. We have used a standard setup that has been used in the evaluation of other models. To evaluate the performances, we split the datasets into the training set, validation set, and testing set with the approximate proportion of 60%, 20%, and 20%, respectively. We use the validation set for hyper-parameter tuning and early stopping and the test set only to report the performance. Throughout all the experiments, we use the *Adam* optimizer with the learning rate as 0.001 and dropout rate as 0.2. The threshold $\lambda_{\mathcal{G}^l}$ of latent connection construction is set to 0.5 as the default. We report F1-micro and F1-macro on the test set of each dataset to evaluate the performance of node classification in the property graphs.

6.4 Code

The code of this work can be downloaded from: https://anonymous.4open.science/r/EPGE-open-code-4C05.

Table 2 Comparison of EPGE with Baselines.

F1-micro(%) / **Dataset** — **Method**	Cora	CiteSeer	PubMed	PPI	HateUser
Publicly Available Implementation of Existing Methods					
DeepWalk	67.20	43.20	65.30	60.66	-
Node2vec	74.90	54.70	75.30	61.98	-
GCN	81.50	70.30	79.00	-	-
GAT	83.22	72.50	79.00	-	-
GraphSAGE-GCN	-	-	-	50.00	-
GraphSAGE-MEAN	-	-	-	59.80	-
GraphSAGE-LSTM	-	-	-	61.20	-
GraphSAGE-MaxPooling	-	-	-	60.00	-
JK-LSTM	85.80	74.70	-	-	-
H-GCN	84.50	72.80	79.80	-	-
N-GCN	83.00	72.20	79.50	46.80	-
DAGNN	84.40	73.30	80.50	-	-
Geom-GCN	84.93	75.14	88.09	-	-
Our Proposed Methods					
EPGE-MEAN	85.80	74.60	**88.43**	65.00	94.20
EPGE-GCN	85.00	74.00	84.50	60.00	92.60
EPGE-LSTM	**86.60**	**75.80**	**88.43**	76.00	92.00
EPGE-MeanPooling	84.60	75.00	87.60	83.00	**92.80**
EPGE-MaxPooling	85.00	74.00	86.97	**85.00**	**92.80**

6.5 Results of Node Classification

In this section, we will compare EPGE with several competing algorithms, and perform a more detailed comparative analysis between EPGE and GraphSAGE.

6.5.1 Comparison with Baselines

We first report node classification accuracy results compared with baselines in Table 2. To ensure a fair comparison, we use the results reported in their respective papers. Therefore, some results are missing because they have not been applied to this collection of datasets. Note, it is common in graph embedding research to use the results on standard datasets, published in original papers, given the experimental setup is consistent across various papers.

From Table 2, we can see that EPGE achieves higher F1-micro scores than all the other methods for all datasets, especially on PubMed and PPI for which performance improvements are significant. The methods that use node property information (i.e., EPGE, GraphSAGE and GCN) achieve better performance than the methods that use the skip-gram model to capture the structure relationships (i.e., DeepWalk and node2vec). In addition, compared with GraphSAGE and GCN-related models (e.g., H-GCN, N-GCN), EPGE further improves the accuracy of node classification, which highlights the effectiveness of our proposed EPGE framework.

6.5.2 Comparison with GraphSAGE

Since our model is conceptually aligned with GraphSAGE, it is necessary to compare with GraphSAGE. To do this, we implement EPGE and GraphSAGE with five

Table 3 Comparison of `EPGE` with `GraphSAGE`.

Models	Cora		CiteSeer		PubMed		PPI		HateUser	
	F1-micro(%)	F1-macro(%)	F1-micro(%)	F1-macro(%)	F1-micro(%)	F1-macro(%)	F1-micro(%)	F1-macro(%)	F1-micro(%)	F1-macro(%)
Our Implementation of GraphSAGE										
GraphSAGE-MEAN	85.20	84.92	73.00	56.63	87.40	87.26	55.00	35.68	92.00	75.08
GraphSAGE-GCN	84.00	83.01	71.20	54.46	82.13	81.68	48.00	24.88	90.40	68.81
GraphSAGE-LSTM	84.00	83.48	72.20	57.60	87.83	87.77	69.00	59.20	88.60	68.61
GraphSAGE-MeanPooling	**85.40**	84.41	74.40	58.48	85.97	86.05	66.00	55.25	90.00	74.86
GraphSAGE-MaxPooling	**85.40**	84.10	72.60	57.54	86.93	86.74	66.00	56.18	67.00	55.64
Our Proposed Methods										
EPGE-MEAN	**85.80**	**86.54**	**74.60**	**59.84**	**88.43**	**88.37**	**65.00**	**52.97**	**94.20**	**81.36**
EPGE-GCN	**85.00**	**85.38**	**74.00**	**58.08**	**84.50**	**84.13**	**60.00**	**43.60**	**92.60**	**74.17**
EPGE-LSTM	**86.60**	**86.89**	**75.80**	**60.05**	**88.43**	**88.33**	**76.00**	**71.02**	**92.00**	**75.08**
EPGE-MeanPooling	84.60	**84.58**	**75.00**	**59.59**	**87.60**	**87.64**	**83.00**	**78.66**	**92.80**	**80.35**
EPGE-MaxPooling	85.00	**84.85**	**74.00**	**59.32**	**86.97**	**86.98**	**85.00**	**81.54**	**92.80**	**78.86**

Table 4 The Results of Ablation Study.

F1-micro(%) **Dataset** / **Method**	Cora	CiteSeer	PubMed	PPI	HateUser
EPGE-MEAN	85.80	74.60	88.43	65.00	94.20
EPGE-MEAN$_{\backslash L}$	86.00(↑ 0.20)	75.40(↑ 0.80)	87.33(↓ 1.10)	69.00(↑ 4.00)	91.40(↓ 2.80)
EPGE-MEAN$_{\backslash B}$	84.60(↓ 1.20)	71.80(↓ 2.80)	87.20(↓ 1.23)	56.00(↓ 9.00)	90.00(↓ 4.20)
EPGE-GCN	85.00	74.00	84.50	60.00	92.60
EPGE-GCN$_{\backslash L}$	84.60(↓ 0.40)	74.80(↑ 0.80)	82.93(↓ 1.57)	65.00 (↑ 5.00)	89.80(↓ 2.80)
EPGE-GCN$_{\backslash B}$	84.60(↓ 0.40)	72.40(↓ 1.60)	82.77(↓ 1.73)	55.00(↓ 5.00)	89.00(↓ 3.60)
EPGE-LSTM	86.60	75.80	88.43	76.00	92.00
EPGE-LSTM$_{\backslash L}$	84.20(↓ 2.40)	74.40(↓ 1.40)	88.00(↓ 0.43)	85.00(↑ 9.00)	89.60(↓ 2.40)
EPGE-LSTM$_{\backslash B}$	82.60(↓ 4.00)	68.80(↓ 7.00)	87.00(↓ 1.43)	65.00(↓ 11.00)	89.40(↓ 2.60)
EPGE-MeanPooling	84.60	75.00	87.60	83.00	92.80
EPGE-MeanPooling$_{\backslash L}$	84.80(↑ 0.20)	74.80 (↓ 0.20)	88.30(↑ 0.70)	87.00 (↑ 4.00)	87.20(↓ 5.60)
EPGE-MeanPooling$_{\backslash B}$	83.40(↓ 1.20)	71.00(↓ 4.00)	87.03(↓ 0.57)	65.00(↓ 18.00)	90.40(↓ 2.40)
EPGE-MaxPooling	85.00	74.00	86.97	85.00	92.80
EPGE-MaxPooling$_{\backslash L}$	84.60(↓ 0.40)	75.60(↑ 1.60)	87.27(↑ 0.30)	86.00(↑ 1.00)	63.80(↓ 29.00)
EPGE-MaxPooling$_{\backslash B}$	84.20(↓ 0.80)	71.80(↓ 2.20)	86.67(↓ 0.30)	59.00(↓ 26.00)	90.20(↓ 2.60)

F1-macro(%) **Dataset** / **Method**	Cora	CiteSeer	PubMed	PPI	HateUser
EPGE-MEAN	86.54	59.84	88.37	52.97	81.36
EPGE-MEAN$_{\backslash L}$	85.79(↓ 0.66)	59.82(↓ 0.02)	87.11(↓ 1.26)	57.85(↑ 4.87)	73.48(↓ 7.87)
EPGE-MEAN$_{\backslash B}$	85.03(↓ 1.42)	56.24(↓ 3.61)	87.25(↓ 1.12)	37.54(↓ 15.43)	55.67(↓ 25.68)
EPGE-GCN	85.38	58.08	84.13	43.60	74.17
EPGE-GCN$_{\backslash L}$	83.55(↓ 1.84)	57.00(↓ 1.05)	82.50(↓ 1.63)	53.39(↑ 9.79)	70.34(↓ 3.82)
EPGE-GCN$_{\backslash B}$	84.65(↓ 0.73)	57.05(↓ 1.00)	82.55 (↓ 1.57)	34.13(↓ 9.46)	47.09(↓ 27.07)
EPGE-LSTM	86.89	60.05	88.33	71.02	75.08
EPGE-LSTM$_{\backslash L}$	83.89 (↓ 3.00)	56.09(↓ 3.96)	87.64(↓ 0.68)	82.39(↑ 11.37)	68.25(↓ 6.82)
EPGE-LSTM$_{\backslash B}$	82.68(↓ 4.21)	53.75(↓ 6.30)	86.80 (↓ 1.53)	49.59(↓ 21.43)	62.71(↓ 12.36)
EPGE-MeanPooling	84.58	59.59	87.64	78.66	80.35
EPGE-MeanPooling$_{\backslash L}$	84.05(↓ 0.54)	59.35(↓ 0.23)	88.16(↑ 0.52)	84.45(↑ 5.79)	69.70(↓ 10.65)
EPGE-MeanPooling$_{\backslash B}$	83.47(↓ 1.12)	55.68(↓ 3.90)	87.05(↓ 0.59)	50.17(↓ 28.48)	63.13(↓ 17.21)
EPGE-MaxPooling	84.85	59.32	86.98	81.54	78.86
EPGE-MaxPooling$_{\backslash L}$	83.85(↓ 1.00)	59.57(↑ 0.25)	87.22(↑ 0.23)	82.18(↑ 0.64)	53.76(↓ 25.10)
EPGE-MaxPooling$_{\backslash B}$	83.90(↓ 0.95)	54.33(↓ 4.99)	86.66(↓ 0.33)	42.84(↓ 38.70)	59.69(↓ 19.18)

different aggregators, namely MEAN, GCN, LSTM, MeanPooling and MaxPooling aggregators. Table 3 summaries the results of GraphSAGE and EPGE for node classification on five public datasets (better results are highlighted in bold).

It is noteworthy that EPGE generally achieves better performance than standard GraphSAGE, especially on PPI and HateUser. Specifically, EPGE achieves higher F1-micro and F1-macro scores than the corresponding GraphSAGE methods except for the Cora dataset. Overall, it is very encouraging to note that our proposed EPGE model can greatly improve the performance of GraphSAGE on the task of node classification in the property graphs.

6.6 Ablation Study

To verify the effectiveness of the proposed latent graph construction and biased sampling strategies, we conduct an ablation study in this section and report both F1-micro and F1-macro as the evaluating metric. In this section, EPGE$_{\backslash L}$ refers to EPGE without latent connections and EPGE$_{\backslash B}$ represents EPGE without biased sampling. The results are shown in Table 4, where the top table presents the F1-micro values and the bottom table reports the F1-macro values. Also, the performance degradation (↓) or improvement (↑) without two strategies of EPGE are given in parentheses.

6.6.1 The Strategy of Latent Graph Construction

We remove the latent graph from the EPGE but preserve the biased sampling strategy. From the results in Table 4, except on PPI dataset, $\texttt{EPGE}_{\backslash\texttt{L}}$ (EPGE without latent graph) generally shows performance degradation than EPGE, with maximum degradation of 29.00% in F1-micro and 25.10% in F1-macro on HateUser. It can be seen that EPGE with certain aggregators yields slightly inferior performance than $\texttt{EPGE}_{\backslash\texttt{L}}$ on Cora, CiteSeer and HateUser datasets (i.e., F1-micro of EPGE-MEAN and EPGE-MeanPooling on Cora dataset, F1-micro of EPGE-MEAN, EPGE-GCN and EPGE-MaxPooling on CiteSeer Dataset, F1-macro of EPGE-MaxPooling on CiteSeer Dataset, F1-micro and F1-macro of EPGE-MeanPooling and EPGE-MaxPooling on PubMed Dataset). However, the performance degradation never exceeds more than 1%. These ablation studies reveal the efficacy of latent connections in learning node embedding especially for datasets where the number of edges is not many.

6.6.2 The Biased Sampling Strategy

To study the impact of biased sampling strategy, we compare EPGE with no biased sampling. It can be seen from the results that the biased sampling strategy outperform $\texttt{EPGE}_{\backslash\texttt{B}}$ (EPGE without biased sampling) on all five datasets. Especially, for EPGE-MaxPooling on PPI dataset, performance improvement can be achieved up to 26.00% and 38.70% in F1-micro and F1-macro, respectively. For HateUser dataset, there are 25.68% and 27.07% performance improvement in F1-macro for EPGE-MEAN and EPGE-GCN, respectively. All experimental results verify the contributions of adopting a biased sampling strategy in learning the node embedding.

6.7 Sensitivity Analysis

In this set of experiments, we evaluate the effects of some important parameters in EPGE on its performance, including the parameter of latent connection construction, the sampling size of neighbours, and the number of edges in the property graph.

6.7.1 The Parameter of Latent Connection Construction

In our proposed model, the latent neighbourhood is determined based on the Pearson similarity. As we discussed, when the similarity between two nodes exceeds the threshold $\lambda_{\mathcal{G}^l}$, they are linked by a latent edge, which finally creates a latent graph. In this section, we probe the influence of the threshold $\lambda_{\mathcal{G}^l}$ on the model's performance. The results are presented in Table 5, where we present the results with two different threshold values i.e., $\lambda_{\mathcal{G}^l}$ is set to 0.5 and 0.8.

Compared with GraphSAGE, EPGE with two different thresholds has better performance on all datasets with just one exception (EPGE-MaxPooling on Cora dataset). A comparison between the performance of $\lambda_{\mathcal{G}^l} = 0.5$ and

Table 5 The Analysis of the Parameter of Latent Connection Construction.

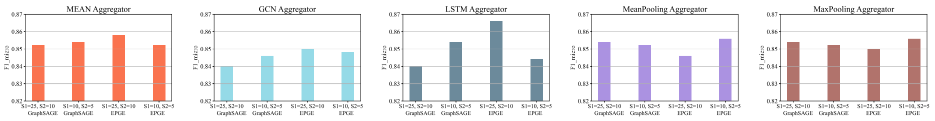
Method	**Parameter**	Cora		CiteSeer		PubMed		PPI		HateUser	
		F1-micro	F1-macro	F1-micro	F1-macro	F1-micro	F1-macro	F1-micro	F1-micro	F1-micro	F1-macro
GraphSAGE-MEAN	-	0.8520	0.8492	0.7300	0.5663	0.8740	0.8726	0.5500	0.3568	0.9200	0.7508
EPGE-MEAN	$\lambda_{\mathcal{G}^l} = 0.5$	**0.8580**	**0.8645**	**0.7460**	**0.5984**	**0.8843**	**0.8837**	**0.6500**	**0.5297**	**0.9420**	**0.8136**
	$\lambda_{\mathcal{G}^l} = 0.8$	**0.8540**	**0.8510**	**0.7420**	**0.5736**	**0.8900**	**0.8896**	**0.6400**	**0.5050**	**0.9300**	**0.8038**
GraphSAGE-GCN	-	0.8400	0.8301	0.7120	0.5446	0.8213	0.8168	0.4800	0.2488	0.9040	0.6881
EPGE-MEAN	$\lambda_{\mathcal{G}^l} = 0.5$	**0.8500**	**0.8538**	**0.7400**	**0.5805**	**0.8450**	**0.8413**	**0.6000**	**0.4360**	**0.9260**	0.7417
	$\lambda_{\mathcal{G}^l} = 0.8$	**0.8580**	**0.8569**	**0.7580**	**0.6089**	**0.8587**	**0.8547**	**0.5800**	**0.4292**	**0.9200**	**0.7652**
GraphSAGE-LSTM	-	0.8400	0.8348	0.7220	0.5760	0.8783	0.8777	0.6900	0.5920	0.8860	0.6861
EPGE-LSTM	$\lambda_{\mathcal{G}^l} = 0.5$	**0.8660**	**0.8689**	**0.7580**	**0.6005**	**0.8843**	**0.8833**	**0.7600**	**0.7102**	**0.9200**	**0.7508**
	$\lambda_{\mathcal{G}^l} = 0.8$	**0.8460**	**0.8372**	**0.7400**	**0.5936**	**0.8877**	**0.8861**	**0.7500**	**0.6833**	**0.9080**	0.6558
GraphSAGE-MeanPooling	-	0.8540	0.8441	0.7440	0.5848	0.8597	0.8605	0.6600	0.5525	0.9000	0.7486
EPGE-MeanPooling	$\lambda_{\mathcal{G}^l} = 0.5$	0.8460	**0.8458**	**0.7500**	**0.5959**	**0.8760**	**0.8764**	**0.8300**	**0.7866**	**0.9280**	**0.8035**
	$\lambda_{\mathcal{G}^l} = 0.8$	**0.8580**	**0.8615**	**0.7480**	0.5794	**0.8903**	**0.8894**	**0.8600**	**0.8277**	**0.9160**	**0.7820**
GraphSAGE-MaxPooling	-	**0.8540**	0.8410	0.7260	0.5754	0.8693	0.8674	0.6600	0.5618	0.6700	0.5564
EPGE-MaxPooling	$\lambda_{\mathcal{G}^l} = 0.5$	0.8500	**0.8485**	**0.7400**	**0.5932**	**0.8697**	**0.8698**	**0.8500**	**0.8154**	**0.9280**	**0.7886**
	$\lambda_{\mathcal{G}^l} = 0.8$	0.8480	0.8392	**0.7480**	**0.5771**	**0.8890**	**0.8891**	**0.8400**	**0.8075**	**0.9260**	**0.7998**

$\lambda_{\mathcal{G}^l} = 0.8$ reveals an interesting pattern. It can be seen that EPGE with MEAN or LSTM aggregators tends to perform better with $\lambda_{\mathcal{G}^l} = 0.5$, whereas EPGE with GCN or MeanPooling aggregators performs better when $\lambda_{\mathcal{G}^l} = 0.8$. We recommend that one should adjust the threshold $\lambda_{\mathcal{G}^l}$ on a validation set instead of setting just one value, as different values are useful for different aggregators as well as datasets.

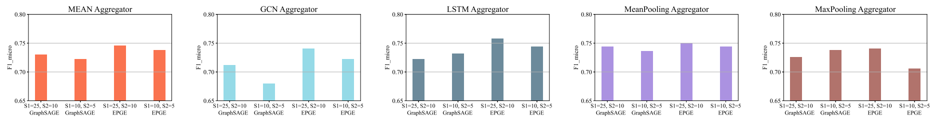
6.7.2 The Setting of the Sampling Size

In this section, we investigate the influence of the size of sampling neighbours on the model performance. [1] recommended the depth of neighbourhood $K = 2$ with neighbourhood sample sizes $S_1 = 25$ and $S_2 = 10$. Here, S_1 and S_2 are the numbers of the sampled neighbours during iteration $k = 1$ and during iteration $k = 2$ of Algorithm 1, respectively. In this experiment, we set the default value for K to 2 and have compared our EPGE model with GraphSAGE by varying the neighbourhood sample sizes $\{S_1, S_2\}$ to two sets of parameters ($\{25, 10\}$ and $\{10, 5\}$). The results on five datasets are presented in Figure 4. For each dataset, we present the F1-micro scores of models with five aggregators. A similar pattern of F1-macro results were observed but not shown due to space constraints. For each aggregator, we present four sets of results, namely, GraphSAGE with parameters $\{S_1, S_2\} = \{25, 10\}$, GraphSAGE with parameters $\{S_1, S_2\} = \{10, 5\}$, EPGE with parameters $\{S_1, S_2\} = \{25, 10\}$, and EPGE with parameters $\{S_1, S_2\} = \{10, 5\}$.

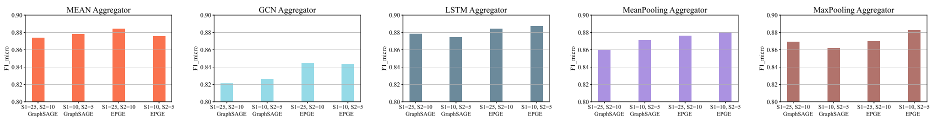
Comparing the results with five aggregators on five datasets, it can be seen that in almost all figures, EPGE with $\{S_1, S_2\} = \{25, 10\}$ generally achieves the best performance, which is consistent with the results in Section 6.5. In addition, whether $\{S_1, S_2\} = \{25, 10\}$ or $\{S_1, S_2\} = \{10, 5\}$ the results of EPGE are generally better than that of GraphSAGE. Now turning to the experimental evidence on EPGE when $\{S_1, S_2\} = \{10, 5\}$ and GraphSAGE when $\{S_1, S_2\} = \{25, 10\}$, we can see that the former has a comparable performance to the latter, or even better, especially on PubMed, PPI and HateUser datasets. It is encouraging to see that a small sampling size for EPGE is able to maintain promising results that was achieved by GraphSAGE with larger neighbours.



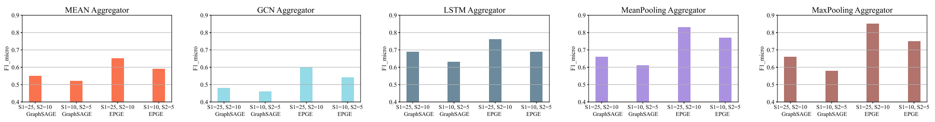
(a) The Setting of the Sampling Size (on Cora Dataset).



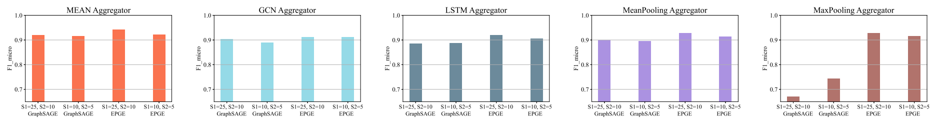
(b) The Setting of the Sampling Size (on CiteSeer Dataset).



(c) The Setting of the Sampling Size (on PubMed Dataset).



(d) The Setting of the Sampling Size (on PPI Dataset).

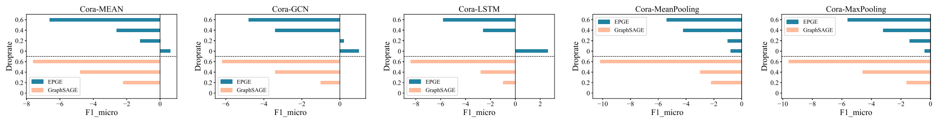


(e) The Setting of the Sampling Size (on HateUser Dataset).

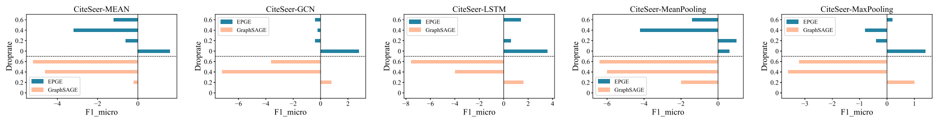
Fig. 4 The Setting of the Sampling Size on Five Datasets

6.7.3 The Influence of the Edges

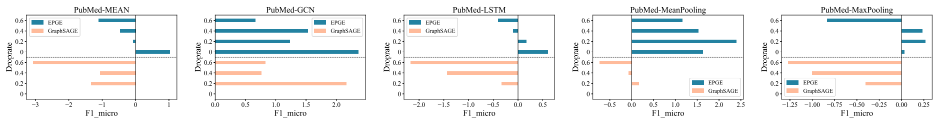
In `GraphSAGE` model, with more edge relationships, nodes can more likely learn informative messages from their neighbourhood, resulting in better performance for node classification. On the other hand, it will face performance deterioration if no sufficient edges are present. Of course, our proposed `EPGE` model can help. In this section, we will evaluate the performance of `GraphSAGE` and `EPGE` when there are very few edges in the property graph. To conduct this experiment, we randomly drop edges at a proportion of 20%, 40%, and 60%, respectively, and then test the models' performance on the adjusted graph. The results on five datasets are shown in Figure 5. We present `F1-micro` scores of `EPGE` and `GraphSAGE` models with five aggregators (The `F1-macro` results show a similar pattern with `F1-micro`). In these results, we use `GraphSAGE` with original edges (i.e., $droprate = 0$) as the baseline, and compare `GraphSAGE` and `EPGE` at $droprate = 0.2$, $droprate = 0.4$, $droprate = 0.6$ with this baseline.



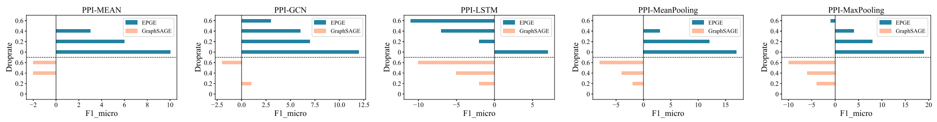
(a) The Influence of the Edges (on Cora Dataset).



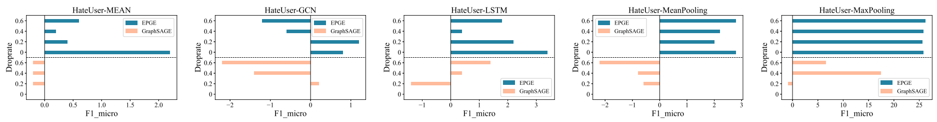
(b) The Influence of the Edges (on CiteSeer Dataset).



(c) The Influence of the Edges (on PubMed Dataset).



(d) The Influence of the Edges (on PPI Dataset).



(e) The Influence of the Edges (on HateUser Dataset).

Fig. 5 The Influence of the Edges on Five Datasets.

The results indicate that `GraphSAGE` generally suffers a performance decrease with higher *droprate* and `EPGE` shows a similar situation. However, when comparing `GraphSAGE` and `EPGE` with the same *droprate*, `GraphSAGE` show worse results, which means that our method outperforms `GraphSAGE` in graphs with much fewer edges, demonstrating the effectiveness of latent connections and bias sampling strategies of `EPGE`.

6.8 Label Consistency Analysis

As presented in Section 5, the label consistency metric τ evaluates the usefulness of the sampled neighbourhood information. Table 6 reports the values of τ in `GraphSAGE` and `EPGE` models on five datasets. It can be seen that `EPGE` has a much higher τ value than `GraphSAGE`, which implies that the node and its sampled neighbours have more of the same labels than that in `GraphSAGE`.

Table 6 The Label Consistency Metric τ of `GraphSAGE` and `EPGE`.

τ **Dataset** / **Method**	Cora	CiteSeer	PubMed	PPI	HateUser
`GraphSAGE`	0.8176	0.7393	0.8144	0.5005	0.7753
`EPGE`	0.8517	0.8084	0.8889	0.9830	0.9982

Based on these results, it can clearly be seen that our proposed `EPGE` has better neighbours selecting and aggregating strategy, and this is the main reason why our method can obtain encouraging performances.

7 Conclusions and Future Work

Most graphs in the real world are property graphs, because other than containing the structure information, rich property information exist for each node in the graphs. The early graph representation learning based on random walk focused only on the structure of graph for learning the node embedding, but overlooked the significance of the properties of nodes. Although `Graph Neural Networks` use the properties as the initial features of nodes and then aggregate feature information of the neighbours, they fail to capture implicit/latent relationships among the nodes, which is implicit in the given structure. To address those limitations in existing methods, we propose a novel framework for property graph representation learning – `EPGE`, which not only exploits the existing graph but also builds a latent graph based on the similarity between nodes in the graph. This new latent connection has the ability to capture the long-distance dependencies from nodes with similar properties but far away in the graph. More importantly, the property graph we construct is simplified into a homogeneous graph, which is simpler and more efficient than complex heterogeneous graphs, hence requiring less memory and computational resources. In addition, `EPGE` has an effective neighbour sampling technique that can choose informative features from neighbours. On five publicly available graph datasets, the proposed model outperforms the state-of-the-art methods including `GraphSAGE` for the task of node classification. We further confirmed the superiority of our proposed formulation through a novel quantitative metric for the usefulness of the sampled neighbourhood in the graph.

Acknowledgements. This work was supported by Strategic Priority Research Program of the Chinese Academy of Sciences with No.XDC02030400, and National Key R&D Program 2021 with No.2021YFB3101001.

Declarations

• Funding: This work was supported by [Strategic Priority Research Program of the Chinese Academy of Sciences with No.XDC02030000], [National Key

Research and Development Program of China with No.2021YFB3101403], and [National Key R&D Program 2021 with No.2021YFB3101001].

- Competing interests: The authors have no competing interests to declare that are relevant to the content of this article.
- Ethics approval: Not applicable.
- Consent to participate: Not applicable.
- Consent for publication: The authors confirm that the manuscript has been read and approved by all named authors and that there are no other persons who satisfied the criteria for authorship but are not listed. The authors further confirm that the order of authors listed in the manuscript has been approved by all of us.
- Availability of data and materials: All data generated or analysed during this study are included in this published article.
- Code availability: The code of this work can be downloaded from: https://anonymous.4open.science/r/EPGE-open-code-4C05.
- Authors' contributions: Conceptualization: [Shu Li], [Nayyar A. Zaidi], [Gang Li]; Methodology: [Shu Li], [Nayyar A. Zaidi]; Software: [Shu Li]; Validation: [Shu Li]; Investigation: [Shu Li]; Writing - Original Draft: [Shu Li]; Writing - Review & Editing: [Nayyar A. Zaidi], [Gang Li]; Supervision: [Nayyar A. Zaidi], [Gang Li]; Formal analysis: [Meijie Du], [Hongfei Zhang]; Validation: [Meijie Du]; Investigation: [Zhou Zhou]; Data Curation: [Hongfei Zhang]

References

[1] Hamilton, W.L., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: NIPS, pp. 1024–1034 (2017)

[2] Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K., Jegelka, S.: Representation learning on graphs with jumping knowledge networks. In: Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018, vol. 80, pp. 5449–5458 (2018)

[3] Li, J., Rong, Y., Cheng, H., Meng, H., Huang, W., Huang, J.: Semi-supervised graph classification: A hierarchical graph perspective. In: The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019, pp. 972–982 (2019)

[4] Manipur, I., Manzo, M., Granata, I., Giordano, M., Maddalena, L., Guarracino, M.R.: Netpro2vec: A graph embedding framework for biomedical applications. IEEE/ACM Transactions on Computational Biology and Bioinformatics **19**(2), 729–740 (2022). https://doi.org/10.1109/TCBB.2021.3078089

[5] Park, N., Kan, A., Dong, X.L., Zhao, T., Faloutsos, C.: Estimating

node importance in knowledge graphs using graph neural networks. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019, pp. 596–606 (2019)

[6] Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings (2017)

[7] Hamilton, W.L., Ying, R., Leskovec, J.: Representation learning on graphs: Methods and applications. IEEE Data Eng. Bull. **40**(3), 52–74 (2017)

[8] Goyal, P., Ferrara, E.: Graph embedding techniques, applications, and performance: A survey. Knowl. Based Syst. **151**, 78–94 (2018)

[9] Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain, pp. 3837–3845 (2016)

[10] Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings (2018)

[11] Platonov, O., Kuznedelev, D., Diskin, M., Babenko, A., Prokhorenkova, L.: A critical look at the evaluation of GNNs under heterophily: are we really making progress? arXiv (2023). http://arxiv.org/abs/2302.11640 Accessed 2023-05-19

[12] Wang, T., Jin, D., Wang, R., He, D., Huang, Y.: Powerful Graph Convolutional Networks with Adaptive Propagation Mechanism for Homophily and Heterophily. Proceedings of the AAAI Conference on Artificial Intelligence **36**(4), 4210–4218 (2022). https://doi.org/10.1609/aaai.v36i4.20340. Accessed 2023-05-19

[13] Newman, M.E.J.: A measure of betweenness centrality based on random walks. Soc. Networks **27**(1), 39–54 (2005)

[14] Fouss, F., Pirotte, A., Renders, J., Saerens, M.: Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. IEEE Trans. Knowl. Data Eng. **19**(3), 355–369 (2007)

[15] Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: online learning of social representations. In: The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014, pp. 701–710 (2014)

[16] Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016, pp. 855–864 (2016)

[17] Perozzi, B., Kulkarni, V., Skiena, S.: Walklets: Multiscale graph embeddings for interpretable network classification. CoRR **abs/1605.02115** (2016)

[18] Chen, H., Perozzi, B., Hu, Y., Skiena, S.: HARP: hierarchical representation learning for networks. In: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th Innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018, pp. 2127–2134 (2018)

[19] Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. In: ICLR (2018)

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

[21] Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. Advances in neural information processing systems **29** (2016)

[22] Duvenaud, D.K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., Adams, R.P.: Convolutional networks on graphs for learning molecular fingerprints. Advances in neural information processing systems **28** (2015)

[23] Atwood, J., Towsley, D.: Diffusion-convolutional neural networks. Advances in neural information processing systems **29** (2016)

[24] Niepert, M., Ahmed, M., Kutzkov, K.: Learning convolutional neural networks for graphs. In: International Conference on Machine Learning, pp. 2014–2023 (2016). PMLR

[25] Hu, F., Zhu, Y., Wu, S., Wang, L., Tan, T.: Hierarchical graph convolutional networks for semi-supervised node classification. In: Proceedings

of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019, pp. 4532–4539 (2019)

[26] Liu, M., Gao, H., Ji, S.: Towards deeper graph neural networks. In: KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020, pp. 338–348 (2020)

[27] Abu-El-Haija, S., Kapoor, A., Perozzi, B., Lee, J.: N-GCN: multi-scale graph convolution for semi-supervised node classification. In: Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence. Proceedings of Machine Learning Research, vol. 115, pp. 841–851 (2019)

[28] Zhu, J., Rossi, R.A., Rao, A., Mai, T., Lipka, N., Ahmed, N.K., Koutra, D.: Graph Neural Networks with Heterophily. Proceedings of the AAAI Conference on Artificial Intelligence **35**(12), 11168–11176 (2021). https://doi.org/10.1609/aaai.v35i12.17332. Accessed 2023-05-19

[29] Du, L., Shi, X., Fu, Q., Ma, X., Liu, H., Han, S., Zhang, D.: GBK-GNN: Gated Bi-Kernel Graph Neural Networks for Modeling Both Homophily and Heterophily. arXiv. arXiv:2110.15777 [cs] (2022). http://arxiv.org/abs/2110.15777 Accessed 2023-05-19

[30] Hou, Y., Chen, H., Li, C., Cheng, J., Yang, M.-C.: A Representation Learning Framework for Property Graphs. In: SIGKDD, pp. 65–73 (2019)

[31] Sen, P., Namata, G., Bilgic, M., Getoor, L., Gallagher, B., Eliassi-Rad, T.: Collective classification in network data. AI Mag. **29**(3), 93–106 (2008)

[32] Namata, G., London, B., Getoor, L., Huang, B.: Query-driven Active Surveying for Collective Classification, 8

[33] Pei, H., Wei, B., Chang, K.C.-C., Lei, Y., Yang, B.: Geom-gcn: Geometric graph convolutional networks. In: International Conference on Learning Representations (2020). https://openreview.net/forum?id=S1e2agrFvS